Universitat de Lleida
Escola Politècnica Superior
Enginyeria Tècnica en Informàtica de Sistemes
Treball de final de carrera

# AMMAF: Agent Mòbil Monitor d'Alteració de Fitxers

Autor: Albert Guim Mas 78096820-k

Director: Francesc Solsona Tehas

Gener del 2007

# Resum

Aquest projecte pretén realitzar un sistema en xarxa que monitoritzi periòdicament l'estat de fitxers clau de cadascun dels computadors de la xarxa emprant el paradigma dels agents mòbils. Es tracta de crear una eina enfocada a la protecció de la xarxa que solucioni problemes típicament associats a les aproximacions existents a les tècniques de la monitorització de fitxers.

El sistema implementat és altament descentralitzat, degut a l'ús de tecnologies de mobilitat de programes. Aquesta característica descentralitza les necessitats de càlcul i també disminueix els requeriments d'infraestructura de comunicacions.

El prototip és un sistema altament portable, degut a l'ús del llenguatge de programació Java i de la implementació del propi sistema pensat per funcionar en un ampli ventall de sistemes operatius. També cal remarcar que ha estat dissenyat de forma modular cosa que, juntament amb la correcta documentació de la seva implementació permetrà efectuar de forma senzilla, modificacions en els aspectes que es desitgin.

Per altra banda, és un sistema altament segur. Això s'aconsegueix posant èmfasis en l'aspecte de la seguretat i la protecció a més d'utilitzar un ampli ventall de models i tècniques criptogràfiques.

# Índex

# Índex de figures

# 1 Introducció

## 1.1 Seguretat i protecció

La seguretat en una xarxa ha estat des de sempre una de les principals preocupacions de nombrosos científics. Multitud d'eines han estat desenvolupades per cobrir la necessitat de dotar de seguretat i confidencialitat a una xarxa, incidint el problema des de molts punts de vista diferents. El problema de la seguretat d'una xarxa es divideix en dos tipus relacionats, seguretat i protecció. Anomenarem seguretat a la robustesa de la xarxa a atacs provinents de fora de la xarxa. Per altra banda, anomenarem protecció a amenaces internes de la xarxa.

Les aproximacions més comuns a la seguretat i la protecció consisteixen en controlar el trànsit de paquets de cada equip de la xarxa, cercant comportaments sospitosos, limitar les llibertats dels usuaris privant-los d'executar tasques que comportin un perill per la seguretat de la xarxa, o dividir la xarxa en seccions i aïllar-les entre si per citar-ne unes quantes de molt esteses.

En el cas d'un organisme o empresa de mida gran, com pot ser la UDL, aquest problema és, si cap, molt més complex. Ens trobem davant una xarxa, que a part de tenir la necessitat de ser segura com totes les xarxes, és d'ús public, amb un propòsit molt genèric i d'una extensió considerable. Els propis usuaris de la xarxa son elements que contribueixen a la seva inseguretat, ja que tenen amplies llibertats i la mida de la xarxa en dificulta el seu control, i els usuaris poden canviar de computador a voluntat. A més és una xarxa que ha de ésser operativa el major temps possible.

Per tant, suposant que un hipotètic atacant aconsegueix superar les defenses de seguretat de la xarxa, pot propagar el seu atac amb molta facilitat, ja que els mateixos usuaris usen diversos computadors. Per altra banda, l'hipotètic atacant, pot ser un usuari normal de la xarxa, ja que la xarxa es d'ús públic. Aquestes peculiaritats donen molta importància al concepte de la protecció.

## 1.2 Estat de l'art

A continuació s'analitzen les solucions existents mes populars que tenen una funció similar al sistema dissenyat, proporcionar protecció a una xarxa.

L'eina de la que s'extreu la idea clau del sistema dissenyat és sgi_fam, un sistema de monitorització de fitxers per entorns Unix o bé ***Gamin***, una simplificació de ***Sgi_fam***[1]. La idea clau és monitoritzar els canvis en els fitxers desitjats per tal de controlar l'estat d'un computador; inclús es pot monitoritzar remotament un sistema, però les similituds s'acaben aquí.

sgi_fam i Gamin estan pensats per actuar en temps real, sense tenir gaire en compte les implicacions i utilitats en seguretat i protecció. Normalment notifiquen canvis en fitxers a altres programes sense suport de seguretat (i.e. notificar al gestor d'arxius la creació d'un fitxer nou per tal de que aquest actualitzi la visió que ofereix al usuari).

No estan pensats per recordar l'estat anterior del sistema ni per mantenir un sistema protegit. Una altra diferència es que la monitorització remota en aquests programes consisteix en una comunicació entre els sgi_fam que resideixen en cada computador implicats. A mes a mes, son eines que només funcionen en sistemes tipus Unix.

Per a entorns Windows existia una eina gratuïta anomenada ***FingerPrint***[2] de l'empresa ***2brightsparks*** que basava el seu funcionament en la mateixa idea del Sistema dissenyat però no disposava de funcionalitat en xarxa ni de mecanismes de seguretat. Desafortunada-ment el seu desenvolupament està parat.

Per altra banda, en l'àmbit de les solucions PKI, existeixen multitud de solucions que proporcionen seguretat a una xarxa de computadors com per exemple ***Ejbca***[9] (funciona amb java) o el ***JSS*** de la fundació mozilla que complirien els requeriments necessaris per donar seguretat i confidencialitat a les comunicacions que es necessiten per al nostre sistema. Per altra banda, si s'han d'implantar només pel correcte funcionament del nostre sistema, requereix d'un computador fiable, ben vigilat i que estigui sempre disponible per tal de poder realitzar qualsevol operació amb seguretat.

Per tant, sgi_fam i Gamin no estan pensats per a oferir protecció ni seguretat i Ejbca únicament proporciona un sistema de PKI que en cap cas compleix la funció d'assegurar el sistema.

Donat que el camp de solucions de seguretat basades en la monitorització de fitxers és un camp poc explotat i menys encara la seguretat en un entorn distribuït, el projecte es desenvolupa en aquest àmbit.

## 1.3 Objectius del projecte

L'objectiu del projecte es construir un sistema en xarxa capaç de monitoritzar periòdicament l'estat de fitxers clau de cadascun dels computadors de la xarxa de forma descentralitzada. D'aquesta manera s'intenten detectar canvis en tots els hosts de la xarxa agilitzant la detecció de hosts compromesos o amb una configuració incorrecta.

Els objectius específics del projecte són els següents:

- Proporcionar un Sistema que permeti monitoritzar canvis en els fitxers dels computadors d'una xarxa de manera descentralitzada.
- Dotar aquest sistema de mobilitat per tal d'evitar l'execució contínua i/o simultània en tots els computadors i evitar el consum de tots els recursos disponibles durant el temps que dura la seva execució.
- Dotar aquest sistema de mecanismes per emmagatzemar i comprovar l'estat dels fitxers desitjats de tota la xarxa.
- Dotar aquest sistema de seguretat i privacitat per tal de evitar que es converteixi en font de problemes o d'informació per un atacant.

- Dissenyar una estructura de l'aplicatiu sòlida i eficient, la qual permeti de forma fàcil la seva modificació i/o ampliació. Cal fer també una bona documentació de l'aplicatiu.

## 1.4 Conceptes teòrics

Per comprendre el funcionament d'aquest sistema s'han de tenir clars un seguit de conceptes criptogràfics i matemàtics teòrics. No es pretén definir de forma completa i formal aquests conceptes. Nomes es pretén introduir al lector en el seu funcionament. Aquests són:

**Clau criptogràfica:** Informació que controla les operacions d'un algorisme criptogràfic.

**Encriptar:** Convertir, usant una clau criptogràfica i un algoritme criptogràfic una informació que es vol protegir d'accessos no autoritzats (anomenada text pla) en informació protegida que no es pot llegir.

**Desencriptar:** Operació contrària a encriptar, usant una clau criptogràfica relacionada amb la utilitzada per encriptar la informació i un algoritme criptogràfic, convertir una informació prèviament protegida en text pla.

**Hash o empremta digital:** Seqüència de bits producte d'una funció unidireccional anomenada funció de hash que converteix una seqüència de bits arbitraria en una altra seqüència de longitud fixa que la identifica. Si dos hashs són iguals es gairebé segur que els ha originat la mateixa seqüència.

**PKI o infraestructura de clau pública:** Infraestructura que permet la autenticació d'identitats, la signatura comprovable de missatges i l'intercanvi d'informació xifrada entre membres autentificats sense haver d'intercanviar informació secreta prèviament. Es basa en cadenes de certificats generats per entitats certificadores en les que es confia.

**Criptografia de clau pública:** La base de la PKI, consisteix en la generació i ús d'una parella de claus criptogràfiques inter-relacionades, una clau privada que s'ha de mantenir secreta i una clau pública que es pot distribuir, que s'empren en algoritmes de xifrat asimètric i en operacions de signatura de dades. Les dades encriptades emprant la clau pública només es poden desencriptar emprant la clau privada associada i viceversa.

**Criptografia de clau privada:** Consisteix en la generació i ús d'una única clau criptogràfica que s'empra en algoritmes de xifrat simètric. Les dades encriptades emprant la clau privada nomes es poden desencriptar emprant la mateixa clau privada. Degut a que el cost computacional de xifrar dades en criptografia de clau pública es molt mes elevat, normalment s'utilitzen esquemes híbrids en els que s'usa la PKI per intercanviar una clau privada que s'emprarà per encritpar les dades que intercanviïn els participants de la conversa.

**Signatura de dades:** Consisteix en generar el hash d'unes dades i utilitzar la clau privada associada a qui ha generat les dades per xifrar aquest hash. Qualsevol receptor de les dades pot calcular-ne el hash i emprar la clau pública de l'emissor per verificar si coincideixen i d'aquesta manera verificar la identitat del emissor.

**Agent mòbil:** Programa informàtic dotat de certa capacitat de decisió en el seu comportament, que té la propietat de poder canviar de computador mantenint parcialment el seu estat d'execució, dotant-lo de mobilitat i capacitat de decisió.

## 1.5 Contingut de la memòria

**Capítol 2:** Ofereix una visió general sobre els agents mòbils. Explica els diversos sistemes d'agents disponibles, com funcionen i com utilitzar-los per sol·lucionar el problema de la protecció en entorns distribuïts.

**Capítol 3:** Proporciona una visió detallada del disseny i la arquitectura del sistema d'agents AMMAF. És en aquest apartat on s'exposen els detalls tècnics i funcionals del sistema. Aquest apartat ha de facilitar la modificació del sistema.

**Capítol 4:** En aquest apartat s'exposa la utilització del sistema d'agents AMMAF des d'un punt de vista pràctic. Aquí s'hi troba des de el procés d'instal·lació fins als esquemes d'ús avançat que és possible implementar.

**Capítol 5:** Aquí s'hi troben les proves de rendiment efectuades, així com un anàlisis de les dades obtingudes al experimentar amb el sistema d'agents.

# 2 Agents mòbils

Havent exposat la situació, ens sorgeix una primera necessitat, escollir una plataforma d'execució per al nostre sistema d'agents que s'adeqüi a les nostres necessitats. Per tant s'analitzaran diversos sistemes existents.

## 2.1 Sistemes d'agents mòbils

En primer lloc, abans de triar una de les diverses plataformes d'agents mòbils cal tenir en compte que existeixen dos estàndards sobre aquest tema, L'Estàndard MASIF i els que empren FIPA[6].

- MASIF

  MASIF, o Mobile Agent System Interoperability Facility es un estàndard creat per l' OMG[1] el 1998 per promoure la interoperatibilitat entre plataformes d'agents. Bàsica-ment, es un estàndard que es centra en especificar els protocols de mobilitat dels agents i en els conceptes que això implica.

- FIPA

  FIPA, o Foundation for Intelligent Physical Agents és una organització que forma part de la IEEE que defineix estàndards tant per la implementació de les plataformes d'agents com també en els actes comunicatius entre agents. L'Estàndard més famós de FIPA és el seu llenguatge FIPA-SL per a la comunicació entre agents mòbils.

Actualment ambdós sistemes estan convergint ja que especifiquen conceptes diferents i són compatibles. A més, FIPA 2000, una de les especificacions de FIPA pretén unificar ambdós sistemes.

Entre tots els sistemes d'agents mòbils, alguns dels més significatius són:

- Concordia: Projecte de Mitsubishi basat en el llenguatge Java. No es gaire popular ja que està aturat des de 2003.
- D'Agents: Projecte Basat en el llenguatge TCL. Està aturat des de 2003.
- aglets: Anteriorment desenvolupat per IBM ara mantingut per la comunitat. És un projecte basat en Java però fa temps que no s'actualitza i no sembla tenir un rumb planejat.
- JADE: Sistema desenvolupat per Tilab basat en Java. És de codi lliure i està essent desenvolupat activament.

[1] Object Management Group

- Jack: Sistema d'agents comercial basat en java. Ofereix moltes facilitats per al disseny de sistemes d'agents.
- Voyager Edge: Sistema d'agents comercial que permet l'ús de varis llenguatges de programació.

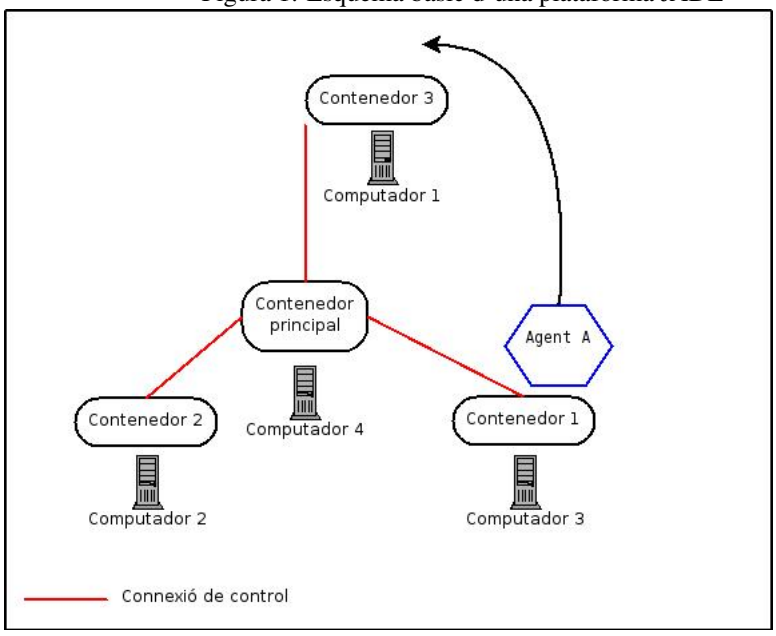
## 2.2 Elecció de JADE

De tots els sistemes anteriors, s'ha escollit JADE ja que està essent desenvolupat activament i disposa d'una bona base d'usuaris. A més, també ofereix facilitats de depuració i control d'agents en temps d'execució. Una altra característica important és que compleix els estàndards de FIPA. Finalment és de codi lliure. A més la simplicitat d'ús i la portabilitat que ens ofereix Java juguen un paper important en la possible portabilitat del sistema d'agents a altres sistemes operatius. En la bibliografia es troba mes informació sobre el funcionament i administració de JADE[15].

Per altra banda, en l'estat actual, el model de seguretat que ofereix JADE es relativament pobre, ja que en entorns on la migració de agents està permesa, no es pot verificar la procedència d'aquests agents ni la seva autenticitat, ni tan sols assegurar les comunicacions amb cap tipus de xifrat. Aquests problemes però, estan essent solucionats amb la inclusió d'un afegit anomenat JADE-S[4]. En la bibliografia es troben diversos anàlisis sobre la seguretat de JADE i de les diverses plataformes d'agents mòbils[18, 14, 19, 13].

El model de Jade consisteix en l'execució en cada màquina d'un o mes contenidors que exerceixen la funció de "hosts virtuals" i permeten la migració des de i cap a ells d'agents mòbils. Aquestes plataformes estant explícitament connectades a un contenidor principal que exerceix la funció de coordinar els contenidors actius informant-los dels agents en execució i els contenidors disponibles. En conjunt la suma de contenidors connectats es denomina plataforma, que es l'entorn per on es poden moure i executar els agents mòbils.

Figura 1: Esquema bàsic d'una plataforma JADE




## 2.3 Plantejament de la solució emprant agents mòbils

La solució proposta consisteix en un sistema d'agents mòbils dissenyats per ser usats amb la plataforma JADE. Aquest sistema d'agents, que es mou de forma segura, emmagatzema l'estat dels fitxers clau de cada computador de la xarxa mitjançant hashs.

Per tal de dotar de confidencialitat i autenticació a totes les operacions que realitza el sistema s'ha implementat també un agent que exerceix les funcions de sistema de PKI.

# 3 Arquitectura i disseny del sistema d'agents

## 3.1 Arquitectura d'AMMAF

AMMAF està format per tres agents mòbils diferents que funcionen cooperativa-ment. Cada un d'ells agents realitza una funció específica. El funcionament conjunt de tots ells permet realitzar les diverses operacions necessàries per actualitzar els hashs que donen informació sobre els hosts de la xarxa. Aquestes funcions són:

**L'agent Signer:** Generació i distribució de parelles de claus certificades.

**L'agent Decrypter:** Control i presentació dels resultats de l'execució del sistema.

**L'agent Updater:** Generació i actualització dels fitxers de hash de cada host.

A la vegada, com ja s'ha dit, tots aquests sistemes funcionen sobre una plataforma JADE.

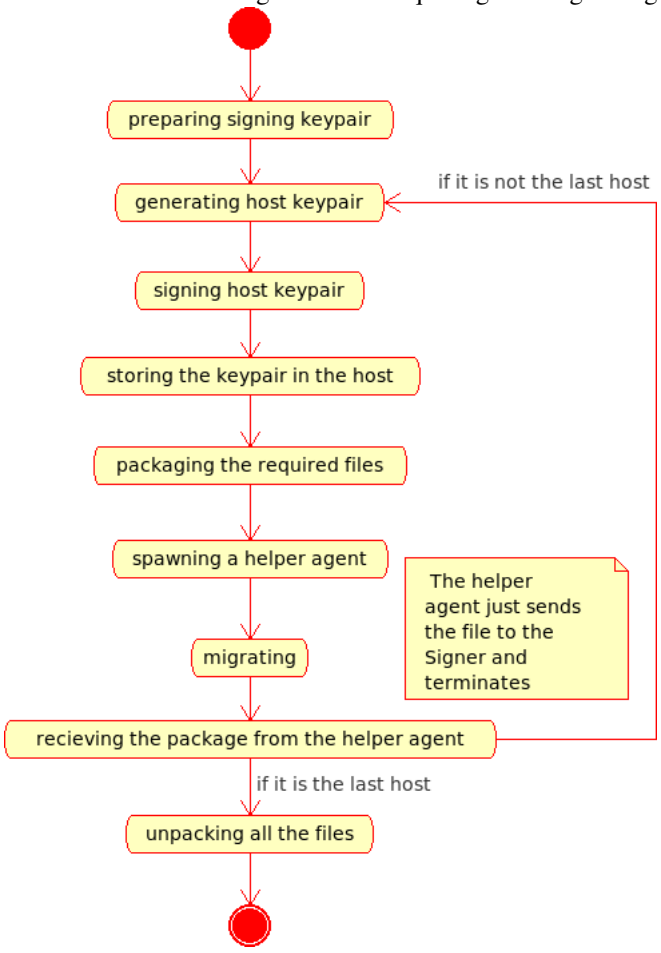
Cal destacar que el control de l'estat dels hosts es realitza comparant els hashs dels fitxers una vegada aquests arriben al host inicial amb els que han estat emmagatzemats prèviament.

### 3.1.1 Agent Signer

Aquest agent, concebut per a una ràpida generació i distribució de parelles de claus certificades, genera una parella de claus en cada host designat com a objectiu. Cada parella de claus, es firma amb una clau mestra que es genera la primera vegada que s'e-xecuta aquest agent i s'emmagatzema al host principal. Per altra banda, aquest agent emmagatzema totes aquestes claus en el host principal que s'empra per comprovar la seguretat de la xarxa. D'aquesta manera la resta d'agents del sistema (i possibles futu-res aplicacions i agents) poden utilitzar aquestes claus per mantenir una comunicació segura sent impossible la suplantació d'identitat per part de hosts maliciosos i també l'espiament de les converses entre agents. Es a dir, dota la plataforma d'agents d'un petit sistema de PKI amb les funcionalitats que això comporta.

Una vegada s'ha generat i certificat una parella de claus per a cada host i aquestes claus han estat emmagatzemades al host principal, es pot tornar a utilitzar aquest agent especificant nous hosts, de manera que els generarà una clau certificada per tal de que es puguin unir al grup de hosts segurs. També es pot utilitzar aquest agent per canviar la clau d'un host segur existent. Es molt important tenir en compte que aquest agent no xifra en cap cas les dades ja que està dissenyat per distribuir ràpidament claus certificades en un entorn on tots els hosts estan controlats, i per tant, s'ha d'utilitzar abans de posar els hosts en un entorn de producció.

Figura 2: Estats que segueix l'agent Signer

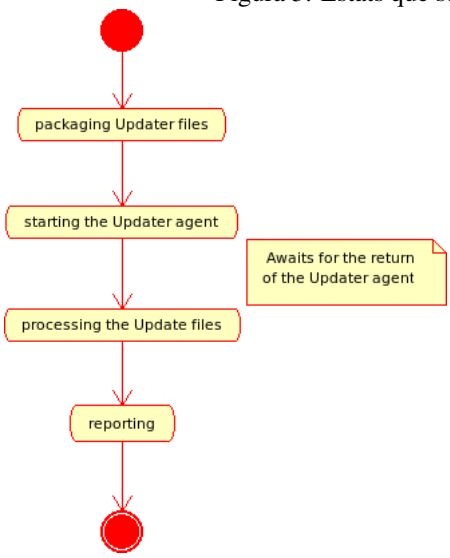



### 3.1.2 Agent Decrypter

Aquest agent, que resideix en el host principal, s'encarrega de executar l'agent que realitza la operació d'actualització (Updater) dels fitxers de hash. En primer lloc empaqueta els fitxers necessaris i genera les claus criptogràfiques necesàries per a que aquests agents puguin funcionar. Una vegada invocat l'agent Updater, simplement es queda a esperar la finalització d'aquest per tal d'informar sobre els resultats de la operació, emmagatzemant de manera ordenada, els diversos arxius del paquet zip que transporta l'agent Updater.

Per tant aquest agent exerceix la funció de preparador i controlador dels agents del sistema.

Figura 3: Estats que segueix l'agent Decrypter

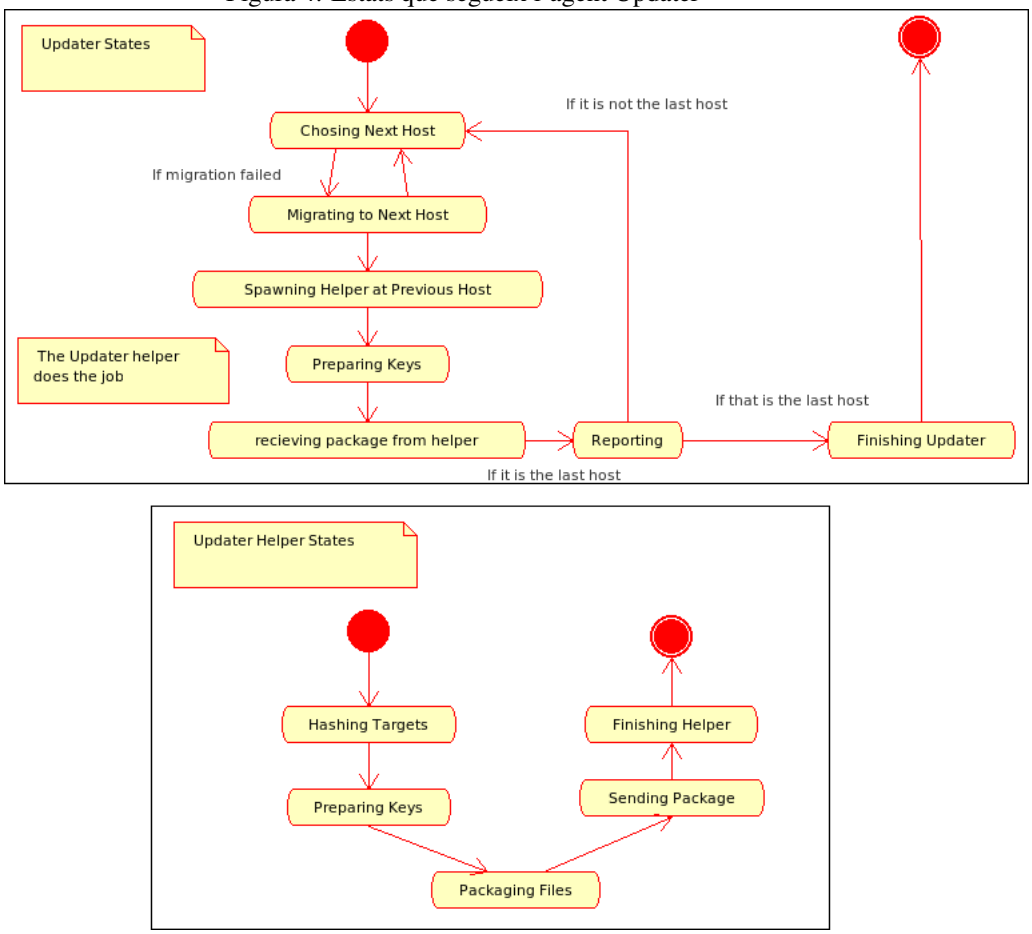



### 3.1.3 Agent Updater

Aquest agent, és invocat per l'agent Decrypter si la operació a realitzar es la de actualitzar els hashs dels fitxers de determinats hosts (mode *update*). Hereta de l'agent Decrypter una clau secreta i xifrada que només pot desxifrar i utilitzar quan aquest agent retorna al host principal per informar, així com també n'hereta la clau pública necessària per verificar que les claus contingudes en cada host estan correctament certificades per l'agent Signer.

Una vegada creat, aquest agent es desplaça per la llista de hosts a actualitzar emprant les diverses claus publiques per transportar de forma segura la informació sensible. En cada host genera una llista de hashs dels fitxers que té especificats en una llista de fitxers, individual per cada host. Una vegada actualitzats els fitxers del host, migra de forma segura al següent objectiu, operació que es repeteix fins que tots els hosts de la llista han estat visitats. Finalment, informa l'agent Decrypter de la finalització de l'operació emprant la clau secreta que aquest li ha llegat.

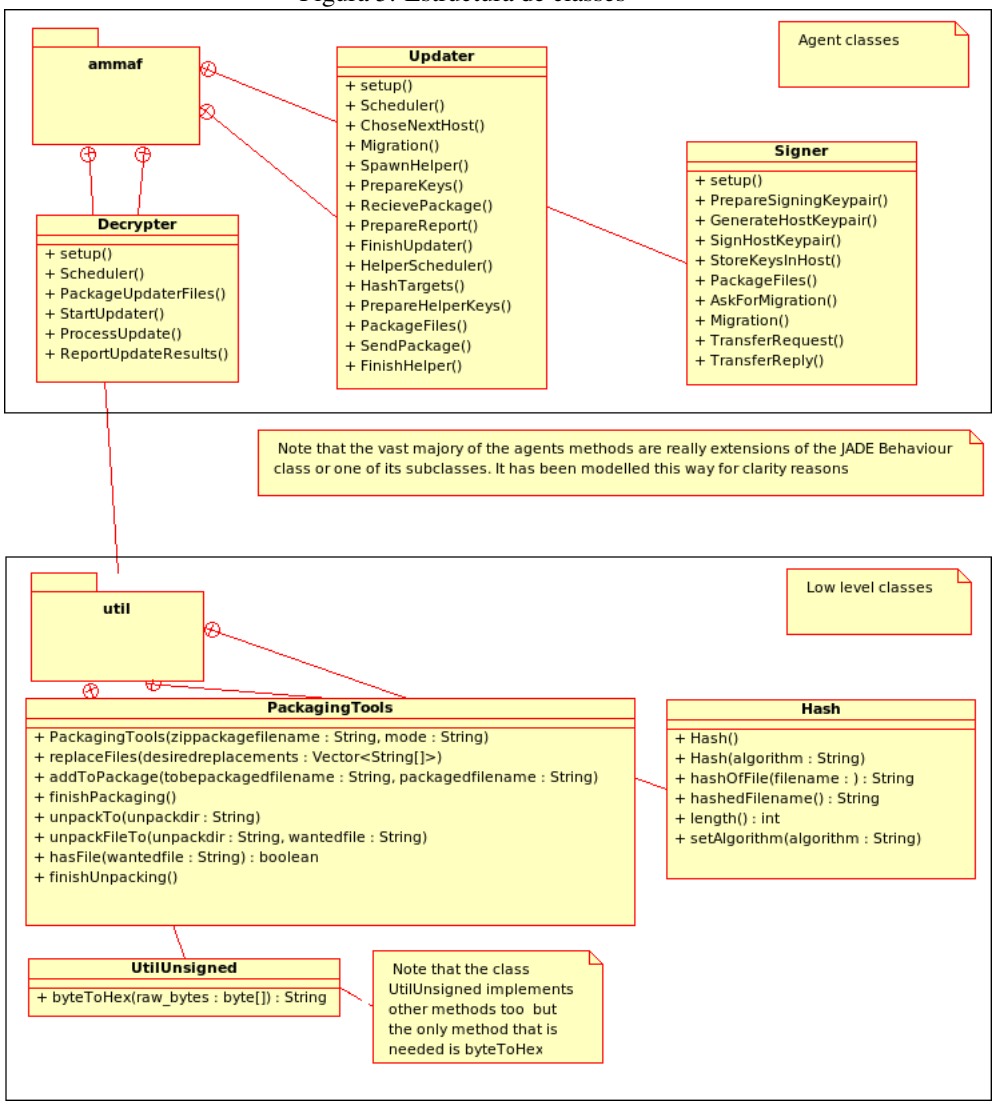
Figura 4: Estats que segueix l'agent Updater




## 3.2 Disseny d'AMMAF

El sistema d'agents, a nivell de disseny està format per tres classes principals que implementen en subclasses, seguint la metodologia establerta per JADE, els comportaments (Behaviours JADE) equivalents als mètodes representats en els diagrames UML adjunts. Es tracta de tres classes homònimes als agents. Per altra banda, aquestes classes utilitzen tres classes auxiliars que realitzen operacions de baix nivell: la classe PackagingTools, la classe Hash i la classe UtilUnsigned.

Figura 5: Estructura de classes




La part de criptografia es basa en la API que ofereix java, JCE[11], així com la generació dels hashs. Per altra banda, la classe auxiliar PackagingTools utilitza la API de Java per empaquetar i comprimir fitxers en format zip.

Cal destacar que les cinc classes dels agents, fan ús de la API de JADE per emprar les capacitats de mobilitat i conformitat amb FIPA.

La conformitat amb FIPA s'aconsegueix mitjançant l'ús d'uns esquemes de conversa en totes les comunicacions entre els diversos agents. Cal destacar que el contingut de

les accions comunicatives, al estar xifrat, no es conforme a les especificacions d'una conversa FIPA formal però es necessari per protegir les dades dels agents. Per aprofundir sobre aquest aspecte veure les especificacions de FIPA[6].

També es important destacar que les converses no utilitzen cap Ontología definida formalment. Això és aixis per evitar la sobrecàrrega que suposa al sistema d'agents comprovar que les converses son correctes, ja que és una funcionalitat adreçada a la programació d'agents intel·ligents en la vessant de la intel·ligència artificial que no és una funcionalitat útil per aquest projecte. No obstant això es possible implementar una Ontología per tal de dotar de capacitats de decisió mes avançades als diversos agents.

Per altra banda, la transmissió dels paquets d'arxius entre hosts, es fa mitjançant la seva divisió i posterior encapsulat en actes comunicatius conformes amb les directrius de FIPA, enlloc de conservar en memòria tots els arxius.

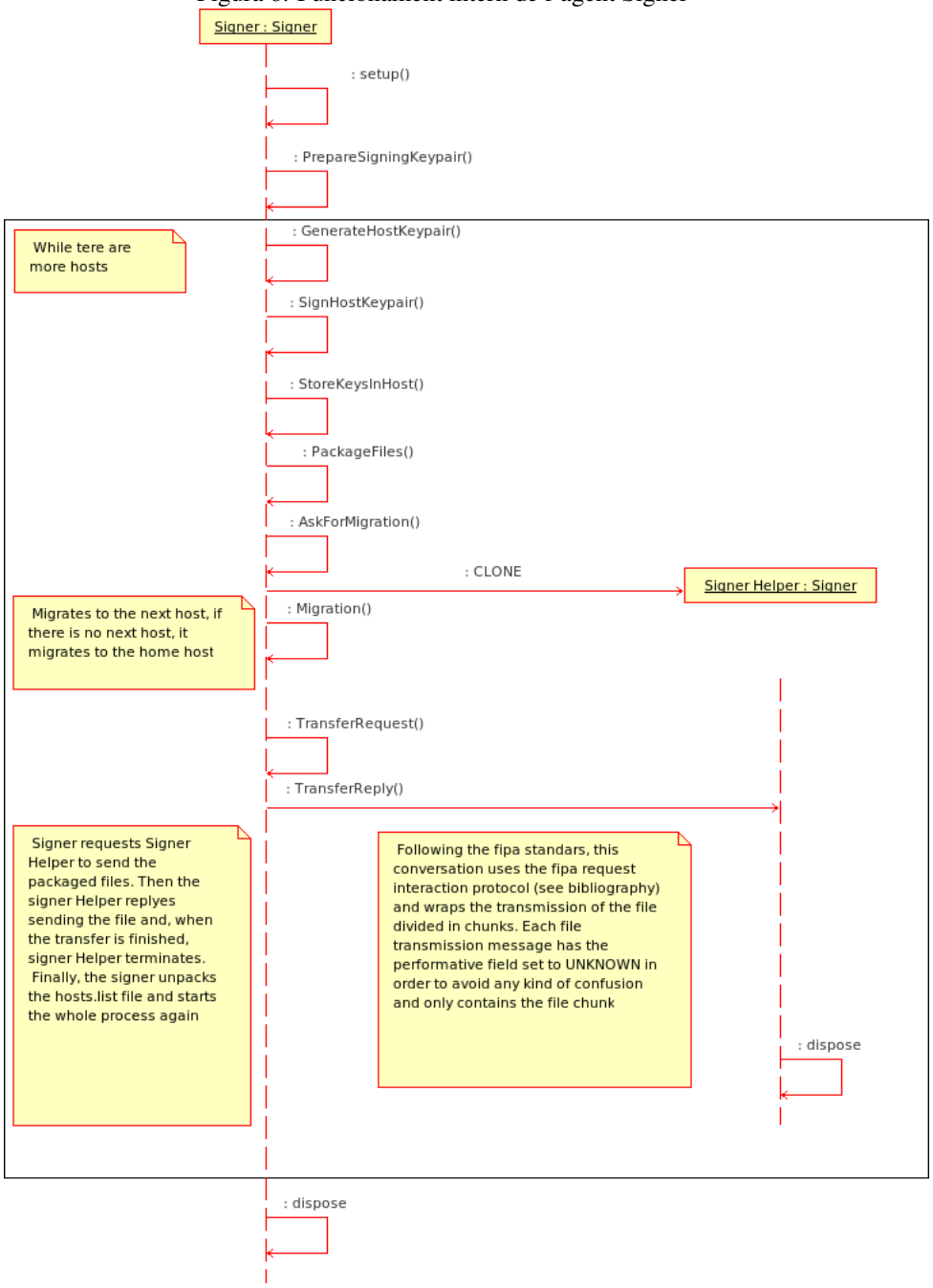
Cal remarcar que, en el tractament de errors, si es detecta algun error en algun fitxer alguna de les firmes o hashs no concorden amb l'esperat o algun agent no es capaç de autenticar-se correctament davant d'un altre, s'eliminen immediatament tots els fitxers implicats i s'interromp la execució de l'operació actual. En els diagrames de missatges, així com en els diagrames que mostren els estats dels agents això no es mostra per raons de claritat i simplicitat.

Finalment, si es vol més informació sobre la funció de cada classe o mètode, a l'annex B es troba la documentació generada per javadoc.

### 3.2.1 Agent Signer

Aquest agent segueix el següent comportament general:

Figura 6: Funcionament intern de l'agent Signer



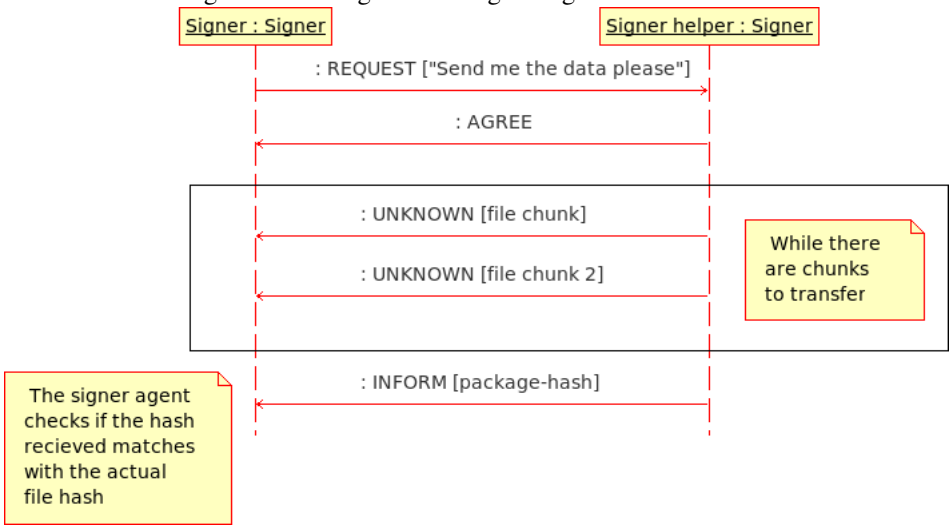

Aquest agent genera dos arxius que contenen la parella de claus RSA usada per certi-

ficar i verificar les claus generades en cada host. La clau privada usada per certificar les claus de cada host s'emmagatzema en format PKCS8 i la clau pública usada per verificar les claus en format X509. Ambdós son formats estàndard de representació de claus criptogràfiques que segueixen la notació ASN.1.

Una vegada generades o carregades en cas de que ja existissin, aquest agent genera una parella de claus RSA en el format esmentat i empra la clau privada general, juntament amb l'algorisme de hash SHA1, per certificar la parella de claus. La firma d'aquestes claus s'emmagatzema en dos fitxers en el host actual.

Posteriorment l'agent empaqueta el fitxer que llista els hosts a visitar, així com el contingut anterior del fitxer, la clau pública que s'acaba de generar i la seva signatura en un fitxer zip, es clona a si mateix i migra al següent host usant les facilitats proporcionades per JADE. L'agent clon, una vegada l'original ha arribat al següent host li transfereix aquest paquet. Per la transferència empra el protocol d'interacció *"Request Interaction Protocol"* definit per FIPA, encapsulant el contingut del fitxer en missatges ACL. Concretament, quan el clon accedeix a transferir el fitxer (missatge agree), envia els missatges amb el fitxer, i quan finalitza, el missatge de notificació (inform-done) conté el hash del fitxer per tal de comprovar la integritat de la transferència.

Figura 7: Missatges entre l'agent Signer i el seu clon




A partir d'aquest punt repeteix el procediment en cada un dels hosts de la llista i quan ha acabat transfereix i desempaqueta el contingut al host inicial.

### 3.2.2 Agent Decrypter

Aquest agent té un disseny senzill. En primer lloc inicia l'execució i si rep el paràmetre update inicia els procediments necessaris per actualitzar els hashs dels hosts de la xarxa.

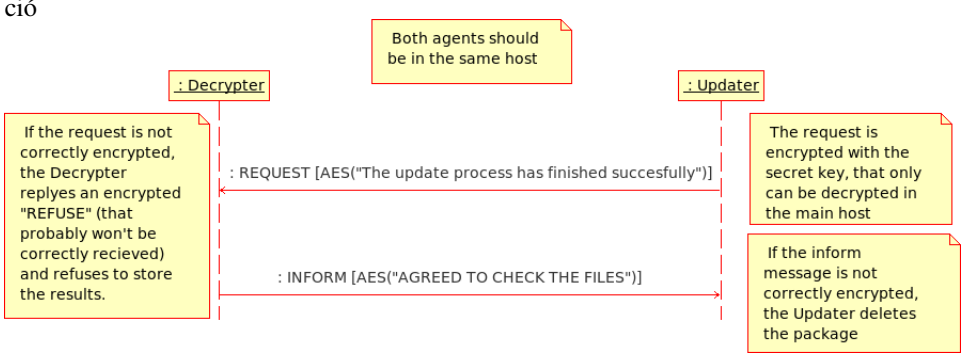
Per fer-ho s'empaqueta el fitxer que llista els hosts a actualitzar, i per cada host, el fitxer que llista els fitxers a actualitzar i també la clau i la signatura de cada un d'aquests hosts en un paquet zip.

En aquest punt prepara els paràmetres per inicialitzar l'agent updater, aquests paràmetres són:

- L'algorisme que s'utilitzarà per generar les firmes (SHA 256 per defecte).
- La clau publica de la autoritat certificadora (la clau pública generada per l'agent signer).
- Una clau secreta (AES) encriptada emprant la clau pública del host en el que resideix l'agent decrypter.

Finalment inicialitza l'agent Updater i s'espera a que aquest agent hagi finalitzat la seva funció i utilitzi la clau secreta que li ha passat per desempaquetar els resultats de la seva operació i donar-los com a vàlids.

Figura 8: Missatges entre l'agent Updater i l'agent Decrypter al finalitzar l'actualització



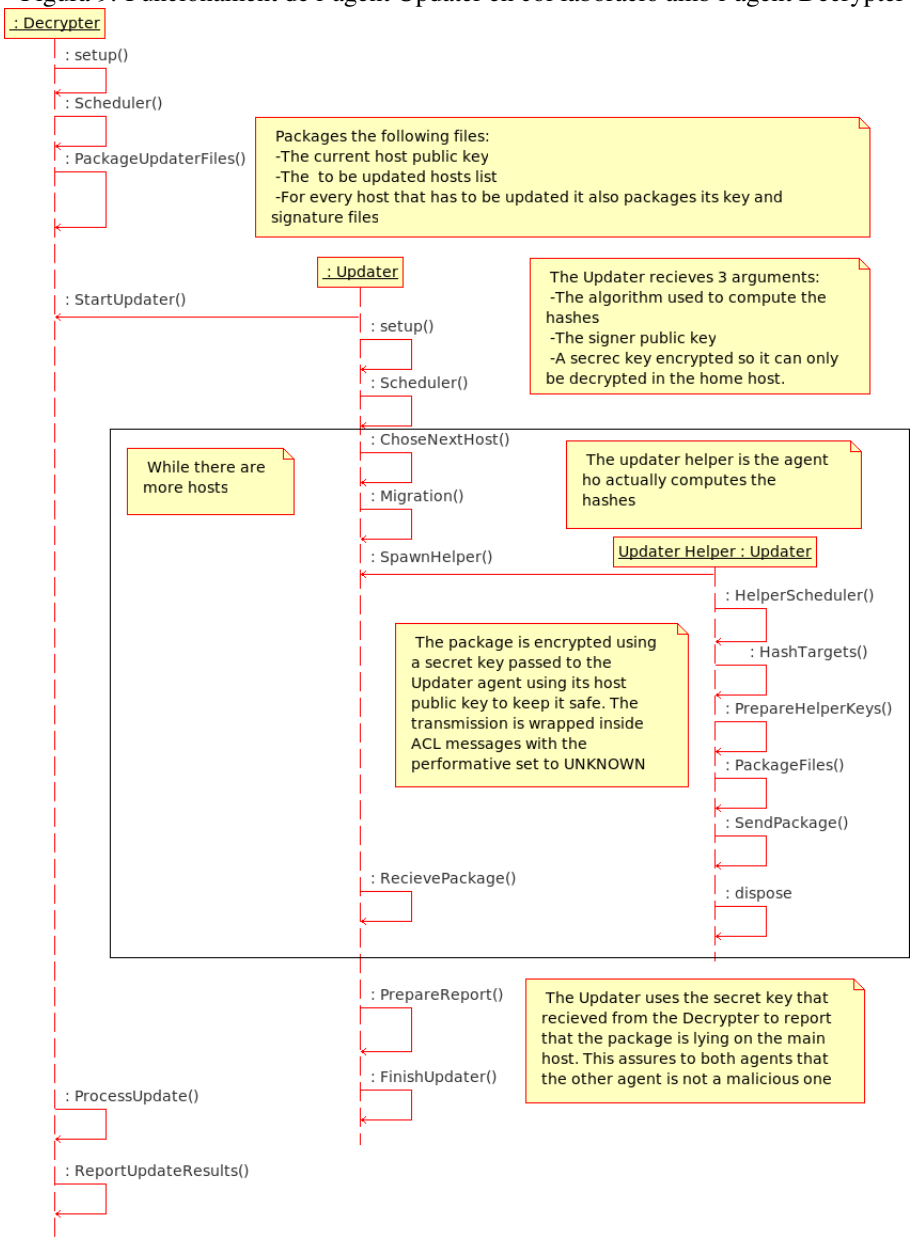

### 3.2.3 Agent Updater

Aquest agent rep de l'agent Decrypter l'algorisme que ha d'utilitzar, la clau pública de l'autoritat certificadora i la clau secreta xifrada, necessària per comunicar-se amb l'agent Decrypter quan l'agent es troba en el host inicial.

L'agent executa sequencialment els següents estats implementats com a comportaments:

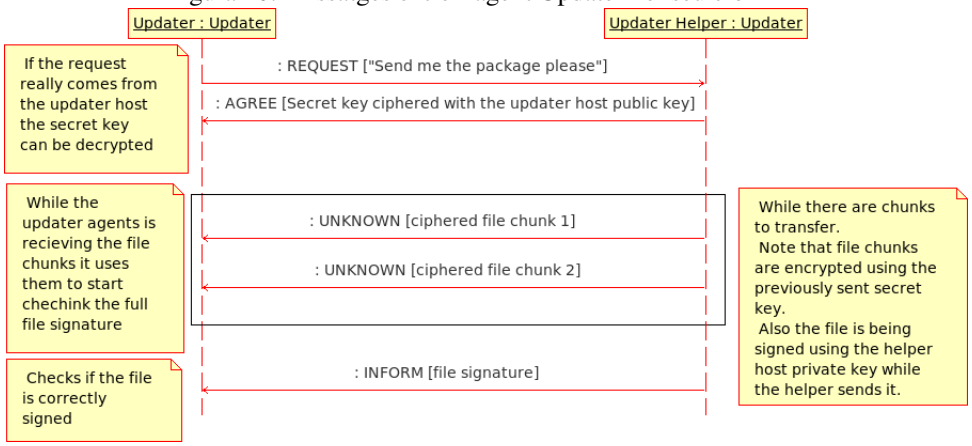
Primerament escull el següent host a actualitzar, i migra emprant les facilitats que ofereix JADE. Una vegada ha arribat correctament al nou host es clona en el host anterior amb la finalitat de transferir els fitxers. Per fer-ho l'agent Updater inicia el protocol *"Request Interaction Protocol"*. Més en detall, el clon genera una clau secreta AES i

Figura 9: Funcionament de l'agent Updater en col·laboració amb l'agent Decrypter



la xifra amb la clau pública del host on està l'Updater, aquesta clau xifrada s'envia com a resposta al missatge que demana la transferència del fitxer. Aquesta clau s'usa per xifrar els missatges que contenen el fitxer zip. Alhora, el clon signa amb la clau privada del host on es troba el paquet, de manera que l'agent Updater, al acabar la transferència en pot verificar l'origen.

Figura 10: Missatges entre l'agent Updater i el seu clon




Aquest procediment es va repetint, amb la diferència que a partir del primer host, l'agent clon també calcula els hashs dels fitxers indicats i actualitza el paquet zip, traient-ne el fitxer que llista els fitxers objectiu per al host al qual es troba i posant-hi el fitxer amb els hashs ja calculats abans d'enviar el paquet a l'agent Updater.

### 3.2.4 Classe Hash

Aquesta classe proporciona les facilitats necessàries per computar els hashs de fitxers. Esta implementada de manera que cada instància sigui re-utilitzable i permeti utilitzar tots els algorismes de hash que suporta Java, així com tots aquells que estiguin implementats seguint les directrius especificades a la JCE[11].

Es important remarcar que fa ús de la classe UtilUnsigned per convertir cadenes de bytes a strings hexadecimals, ja que no existeix cap facilitat per fer-ho en les classes de Java que proporciona Sun Microsystems.

### 3.2.5 Classe UtilUnsigned

Aquesta classe, implementada per Davis Swan i alliberada en llicència GNU LGPL s'encarrega de convertir cadenes de bytes a cadenes hexadecimals per representar els hashs. Per fer-ho s'utilitza el mètode *"byteToHex"*. També es capaç de convertir entre altres diversos tipus de dades. Si es vol aprofundir sobre el funcionament d'aquesta classe examineu el seu codi font, ja que es senzill i ben documentat.

### 3.2.6 Classe Packagingtools

Aquesta classe s'encarrega de totes les operacions de creació, modificació i desempaquetament de paquets zip. Està pensada per executar les operacions bàsiques de maneig de fitxers comprimits, així com la substitució de fitxers en un paquet zip. Tot això sense abusar de la memòria disponible ni fent un gran nombre d'escriptures al disc. Els noms dels mètodes son auto-explicatius, si tot i amb això, a l'annex B es pot trobar la documentació javadoc.

# 4 Manual d'usuari

## 4.1 Instal·lació del sistema

Una bona implantació d'un sistema d'agents requereix instal·lar i configurar JADE en la xarxa de computadors que es desitgi monitoritzar. A més, els fitxers del sistema d'agents no han de ser accessibles ni modificables per un usuari maliciós. Per aconseguir això es recomanable aïllar els fitxers de la plataforma Java i del sistema d'agents de la resta de fitxers del propi sistema operatiu. Els principals mecanismes per aconseguir-ho són:

**Virtualització del Sistema usuari:** Mitjançant l'execució en una màquina virtual del sistema operatiu al que té accés l'usuari, es fa impossible que aquest tingui accés als fitxers de la plataforma, que estarien situats en el sistema operatiu host. Es pot utilitzar User Mode Linux[7], VMWARE[8] o qualsevol software que permeti la creació de màquines virtuals.

**Engabiament chroot:** Similar a l'anterior opció però sense virtualitzar cap sistema operatiu. Es tracta de restringir l'accés a una part del sistema de fitxers de manera que la plataforma i tots els seus requeriments estiguin fora de l'abast dels usuaris.

Aquests procediments, els detalls dels quals queden fora de l'objectiu d'aquest projecte, garanteixen que els fitxers de la plataforma siguin inaccessibles per un usuari maliciós.

En aquest punt, es necessita disposar d'una instal·lació funcional de Java (al menys la versió 1.4) i instal·lar la plataforma JADE amb l'afegit JADE-S. El paquet JADE-S encara que està en desenvolupament i no es completament operatiu, pretén dotar a JADE d'autenticació i autorització integrada amb el sistema operatiu així com permetre la transmissió segura d'informació emprant criptografia.

Per fer-ho s'han de descomprimir els fitxers de Jade en qualsevol directori i afegir el subdirectori *lib* a la variable CLASSPATH per tal de que Java pugi localitzar i utilitzar JADE.

S'ha de descomprimir també el paquet JADE-S en el subdirectori add-ons, i afegir el subdirectori *add-ons/security/lib* a la variable CLASSPATH.

Llavors, s'afegeixen els serveis de seguretat desitjats al fitxer de configuració de les opcions d'arranc de la plataforma (secció services), es configuren els paràmetres criptogràfics i es tria el sistema d'autenticació. D'aquesta manera al engegar JADE un quadre de diàleg comprova si la autenticació es correcta, i els permisos dels agents es redueixen.

En aquest punt es configura el fitxer de polítiques, donant els permisos desitjats a cada usuari i agent, donant permís de lectura als fitxers i directoris que es vulguin monitoritzar així com permís de escriptura al directori de treball del sistema de monitorització.

Per acabar, depenent de si el computador és el controlador del sistema o és només un computador destinat als usuaris es copien els programes monitor necessaris en una carpeta afegida al CLASSPATH. En cas de que el computador sigui el controlador del sistema es requereix copiar totes les classes. En canvi, si es un computador destinat als usuaris, es pot obviar el Decrypter.

Per acabar amb la instal·lació bàsica, si es desitja aprofundir en la metodologia de configuració de la plataforma i els permisos de seguretat adreceu-vos a la bibliografia de JAAS[12] així com la documentació sobre l'administració de JADE[15] per conèixer com configurar els paràmetres avançats de la plataforma.

## 4.2 Funcionament bàsic del sistema amb exemples d'ús

El sistema està format per tres agents mòbils, que requereixen una plataforma JADE, formada per la interconnexió dels contenidors en cada un dels hosts (un d'ells fa la funció de contenidor principal per la plataforma JADE). Aquests tres agents realitzen dues operacions bàsiques; la generació i certificació de noves parelles de claus i la generació o actualització dels fitxers de hash.

Una peculiaritat del sistema, es que el host que realitzi la funció de host principal no ha d'ésser forçosament el host que està executant el contenidor principal. Per altra banda JADE pot funcionar mitjançant una interfície gràfica (emprant l'argument -gui al iniciar qualsevol dels contenidors), que no es res més que un agent gràfic, o des de la consola.

Per executar el contenidor principal s'utilitza la comanda:

```
$ java jade.Boot
```

Aquesta comanda engega un contenidor principal de JADE en el host actual, usant com a directori de treball el directori actual.

Per altra banda, un host que es vulgui incorporar a la plataforma ha d'executar la comanda:

```
$ java jade.Boot -host beherith -container
```

Per convenció, anomenarem host ammaf el host principal on s'emmagatzemen i es llegeixen els fitxers necessaris per iniciar i finalitzar cada una de les operacions bàsiques i directori ammaf al directori de treball on s'invoca la plataforma JADE en cada host.

### 4.2.1 Generació de noves parelles de claus

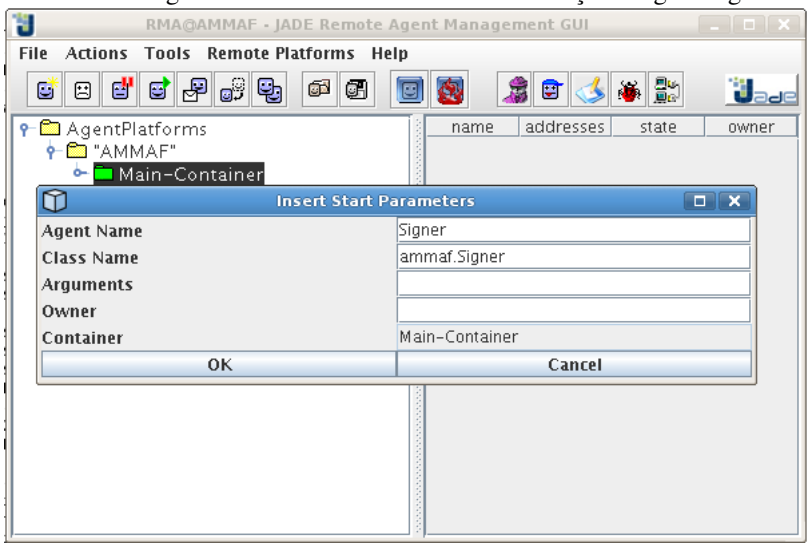
Per generar noves parelles de claus en cada host, en primer lloc necessitem disposar d'un fitxer de hosts objectius correctament construït. El format del fitxer de hosts es

senzill, es un fitxer de text que conté noms de host (en realitat el nom dels contenidors de cada host) separats per retorns de carro. El nom del fitxer és *"hosts.list"* i ha d'estar ubicat en el host ammaf, en el directori ammaf. Un exemple senzill amb dos hosts (en aquest exemple es tracta del host principal i un host d'usuari) seria:

```
beherith
AMD64DC
```

Una vegada s'ha creat i omplert aquest fitxer amb els noms dels hosts als que es vol generar una parella de claus, simplement s'utilitza el RMA, la interfície gràfica de control i monitorització de JADE, per invocar l'agent **Signer.**

Figura 11: Interfície RMA de JADE llençant l'agent Signer




Aquest agent fa la funció de generar, certificar i emmagatzemar les parelles de claus en cada host, així com emmagatzema una còpia de la clau pública de cada un dels hosts. Aquesta clau l'emmagatzema en un fitxer anomenat "<nom del host>.key" i la seva certificació, en un fitxer anomenat "<nom del host>.sig" en el host fam, concretament el directori *"keys"* ubicat dins el directori ammaf. Per acabar emmagatzema la parella de claus utilitzada per signar la resta de claus en el directori ammaf, en dos arxius anomenats *"master_pubkey.key"* i *"master_prikey.key"*.

### 4.2.2 Generació o actualització de nous fitxers de hash

Per generar o actualitzar nous fitxers de hash, necessitem disposar d'un fitxer de hosts objectius correctament construït, amb el mateix format que el fitxer utilitzat per generar

les parelles de claus, i situar-lo en el host ammaf, en el directori ammaf. El nom d'aquest fitxer és *"updatehosts.list"*.

Per altra banda, es necessita un fitxer que llisti els fitxers objectius per a cada host. Aquests fitxers, anomenats amb el nom del host amb extensió .targets, ha de contenir el fitxers dels que es vol obtenir els hash. Aquests fitxers estaran situats en el host ammaf, en el subdirectori *"targetfiles"*. Un exemple de l'estructura d'aquest tipus de fitxer seria, per al host *"AMD64DC"*, un fitxer anomenat *"AMD64DC.targetfiles"* que contindria el següent:

```
/home/gatsu/java.bin
/home/gatsu/jade/lib/jade.jar
/home/gatsu/jade/lib/jadeTools.jar
```

Per acabar els preparatius per generar nous fitxers de hash, necessitem haver executat prèviament l'agent Signer per tal de tenir una parella de claus a cada host, així com els fitxers amb les claus públiques i les seves signatures en el host ammaf.

Una vegada disposem del fitxer de hosts objectius, dels fitxers de fitxers objectius per a cada host i a més tenim els fitxers de claus generats en tots els hosts en els que volem generar o actualitzar llistes de hashs, s'utilitza el RMA per invocar l'agent **Decrypter** amb l'argument *"update"*. Aquest agent, prepara tots els fitxers i informació necessària per a llençar l'agent **Updater**, que es qui s'encarrega de generar els fitxers de hash en cada host i espera els resultats en el host fam. Finalment, quan l'agent **Updater** notifica a l'agent **Decrypter** que ja ha acabat el procés, finalitza la seva execució i aquest accepta els hashs com a vàlids, dipositant-los en el subdirectori *"hashes*" del host ammaf. Aquests fitxers de hash, anomenats amb el nom del host amb extensió .hashs contenen una cadena de text hexadecimal que representa el hash i el caràcter d'espai seguit del nom del fitxer objecte del hash. Seguint amb l'exemple anterior, el fitxer resultant Per al host *"AMD64DC*" suposant un fitxer *"AMD64DC.targetfiles*" com l'anterior-ment esmentat, el fitxer *"AMD64DC.hashes"* resultant seria de la forma següent:

```
C24B3A5EAA52EAE6A1564DC845352828D8BFDB3C7FB25BA863CEF2D2D772048C /home/gatsu/java.bin
609E08175C0C6BA7C3C26D924418CB83EEF7CA9DD6BE1CC53BAC84012F8D1675 /home/gatsu/jade/lib/jade.jar
4A6018B98413F29BA1506641F0BB28A0A8AC8508EF8E6ED6BF989D86686E1999 /home/gatsu/jade/lib/jadeTools.jar
```

### 4.2.3 Comprovació de l'estat dels hosts

La comprovació de l'estat dels hosts es realitza d'una forma molt senzilla. Només cal emmagatzemar els fitxers de hash obtinguts al executar l'agent Decrypter per actualitzar els fitxers de hash i guardar aquests fitxers en un directori segur.

Quan es desitgi comprovar si l'estat dels hosts ha canviat, es torna a executar l'agent **Decrypter** en mode "*update*" per obtenir nous fitxers de hash. Finalment, emprant alguna eina de comparació de fitxers, com diff, podem veure els fitxers que han estat modificats desde la última actualització.

És molt important remarcar que la seguretat que ofereix el sistema està lligada a l'encert a l'hora de triar els fitxers a monitoritzar. AMMAF només proporciona l'entorn per controlar fitxers de forma centralitzada i queda a discreció de cada administrador de xarxa decidir quins fitxers mereixen ésser monitoritzats.

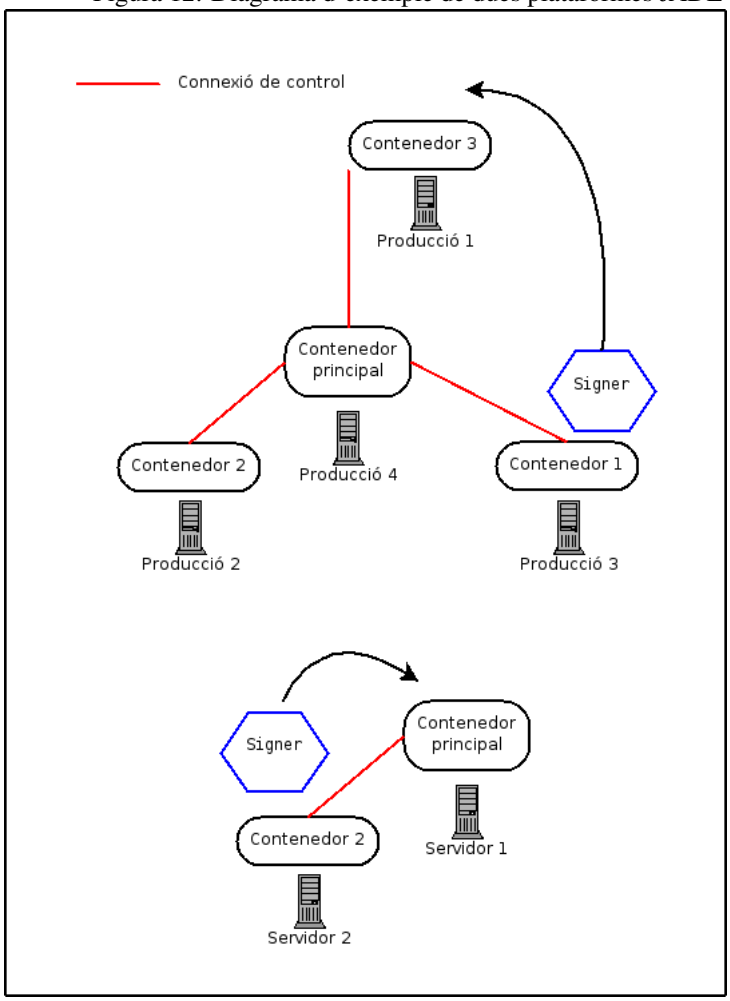
## 4.3 Esquemes de funcionament avançats

### 4.3.1 Divisió de la xarxa en zones aïllades

Un dels possibles esquemes de funcionament a l'hora d'implementar el sistema en una xarxa real pot ésser dividir la xarxa en grups de hosts inaccessibles entre si. Aquesta aproximació és útil en el cas de que es vulguin separar els computadors d'una xarxa depenent de la seva funció seguint criteris de seguretat. Una decisió així pot ésser motivada per exemple, per la necessitat d'aïllament entre els agents que circulen pels computadors que allotgen els servidors accessibles des de l'exterior de la xarxa respecte als computadors de producció interns de la xarxa.

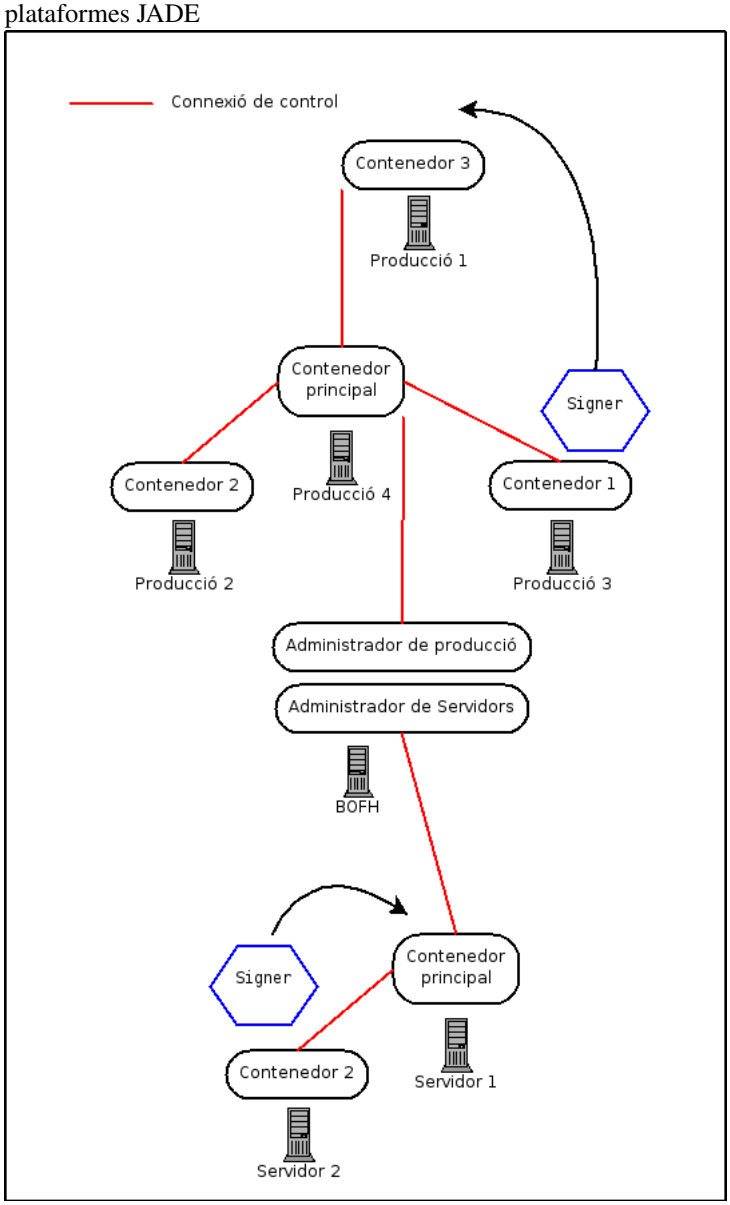
Per abarcar aquest supòsit, es poden utilitzar mes d'una plataforma JADE, de manera que els contenidors de cada host pertanyin a la plataforma que correspon a la seva zona i, per tant, no pugui enviar agents ni interferir amb la plataforma de les altres zones.

Figura 12: Diagrama d'exemple de dues plataformes JADE separades




Aquesta aproximació es molt més flexible, ja que un mateix computador pot tenir mes d'un contenidor i, per tant, pot realitzar la funció de host fam per a més d'una zona. Per aconseguir aquesta funcionalitat només es requereix que els diversos computadors que estiguin en un mateix host, utilitzin un port TCP diferent per a comunicar-se amb els diversos contenidors de les seves respectives plataformes. Veure capítol 2.3 en[15] per a més informació sobre com configurar les plataformes JADE.

Figura 13: Diagrama d'exemple d'una xarxa amb un computador pertanyent a dues plataformes JADE

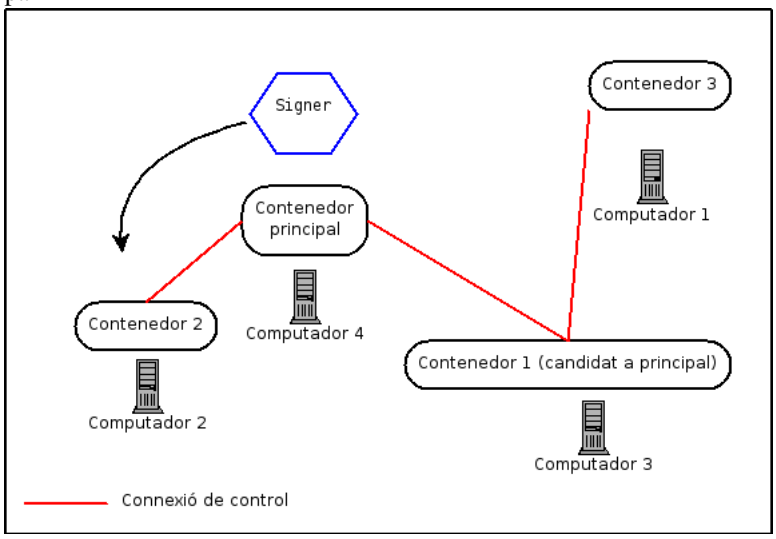



### 4.3.2 Robustesa mitjançant la replicació de contenidors principals

Un altre possible esquema de funcionament a l'hora d'implementar el sistema en una xarxa real consisteix en replicar el contenidor principal d'una plataforma JADE per

incrementar la tolerància a fallades del sistema. Esencialment consisteix en tenir contenidors a altres hosts que també executin la funció de candidats a contenidor principal. D'aquesta forma si el contenidor principal deixa d'estar operatiu, un d'ells assumeix les seves funcions. Aquesta aproximació és molt útil ja que permet que, en cas de que el contenidor principal de JADE caigui, la plataforma no deixi d'estar operativa.

Figura 14: Diagrama d'una plataforma amb contenidors candidats a contenidor principal




En aquest exemple, si el contenidor principal cau, el contenidor 1 passarà a exercir de contenidor principal i el contenidor 2 es connectarà al contenidor 1.

Aquest esquema de funcionament es compatible amb la divisió de la xarxa en diverses plataformes JADE, de manera que cada plataforma pot tenir varis contenidors candidats a contenidor principal, augmentant així la tolerància a fallades del sistema.

Per obtenir mes informació sobre la configuració de plataformes resistents a fallides, consultar el manual d'administració de la plataforma JADE[15].

### 4.3.3 Implantació de restriccions en els permisos dels agents mòbils

Aquesta funcionalitat essencial, consisteix en modificar la política de permisos d'agents en cada host de manera que no puguin llegir o modificar determinats fitxers. Això ens proporciona més seguretat ja que només es requereix permís d'escriptura en els directoris on s'executa la plataforma, en canvi, la resta del sistema de fitxers pot estar protegit contra escriptura, permetent només la lectura dels directoris i fitxers que es vulguin supervisar.

El principal avantatge d'aquest esquema de funcionament radica en que, en el supòsit de que un atacant aconsegueixi utilitzar la plataforma JADE com a vector d'atac a la xarxa, no podrà modificar els fitxers que afecten el correcte funcionament dels computadors.

Per altra banda, es poden implementar configuracions en base a diferents usuaris, de manera que els agents que provinguin de diferents usuaris tinguin permisos diferents segons es requereixi. Reduint d'aquesta manera els riscs en cas de que la plataforma JADE tingui un ús compartit.

La notació utilitzada per configurar els permisos es la mateixa que la que empra JAAS, vegeu-ne el manual de configuració [12].

# 5 Experimentació

Per tal de realitzar un anàlisis sobre la sobrecàrrega i el cost computacional del sistema d'agents AMMAF s'ha fet un anàlisis mesurant el temps que triga al sotmetre el sistema a situacions de funcionament extremes. Aquestes situacions són les següents:

- Mesurar el rendiment del sistema AMMAF quan es veu obligat a transmetre una gran quantitat de dades entre els hosts qie ja de verificar.
- Mesurar el rendiment del sistema AMMAF quan es veu obligat a calcular hashs d'una gran quantitat de dades en cada host.

En ambdós casos, el factor dominant d'aquest rendiment es l'agent Updater, que es el que realitza la major part de les funcions.

Finalment s'ha quantificat el temps que triga l'agent Signer en executar la seva funció en un nombre reduït de hosts.

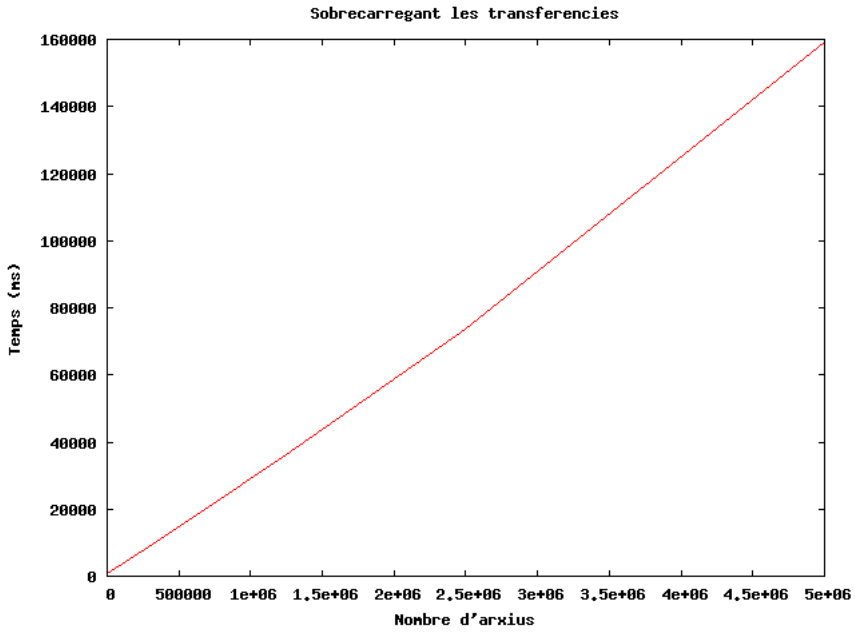
## 5.1 Estudi del rendiment al transmetre una gran quantitat de dades

Per mesurar el temps que triga l'agent Updater en transmetre una gran quantitat de dades se l'obligarà a calcular el hash d'una gran quantitat de fitxers buits. D'aquesta manera, el cost computacional de calcular un hash es mínim de manera que aconseguim minimitzar la interferència temporal que causa, però les dades a comprimir i transmetre al següent host adquireixen un gran volum.

A continuació s'exposa el temps en mil·lisegons que tarda el sistema a comprovar un determinat nombre de fitxers buits en un host i informar al host ammaf:

| | 1 fitxer | 312500 fitxers | 625000 fitxers | 1250000 fitxers | 2500000 fitxers | 5000000 fitxers |
|---|---|---|---|---|---|---|
| mida targetsfile | 51B | 8.6MB | 17.3MB | 34.6MB | 69.1MB | 138.3MB |
| mida hashesfile | 116B | 28MB | 56MB | 112.1MB | 224.1MB | 448.2MB |
| Mostra 1 | 591 | 9494 | 18140 | 37970 | 78756 | 159093 |
| Mostra 2 | 737 | 9497 | 18096 | 35632 | 74054 | 159339 |
| Mostra 3 | 779 | 9457 | 18617 | 35481 | 72268 | 159001 |
| Mostra 4 | 695 | 9397 | 17993 | 35422 | 69810 | 158735 |

Figura 15: Evolució del rendiment al transmetre una gran quantitat de dades




Com es pot observar a la gràfica es tracta d'un rendiment gairebé lineal, una propietat que es favorable per tal d'implementar el el sistema a una escala més gran. Per altra banda, a la taula es pot observar que l'estat de la xarxa juga un paper molt important en transferències amb un baix volum de dades.

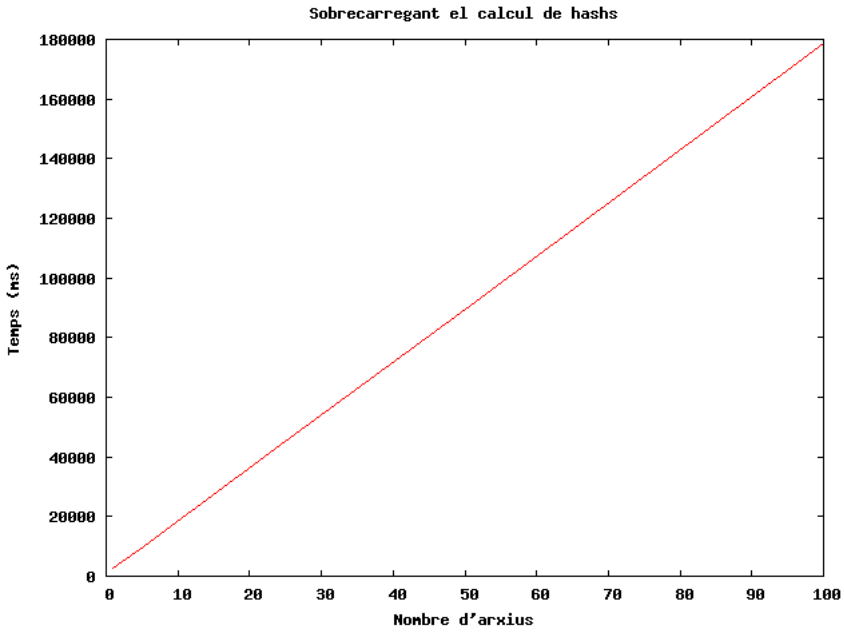
## 5.2 Estudi del rendiment al processar una gran quantitat de dades

Per mesurar el temps que triga l'agent Updater en processar una gran quantitat de dades se l'obligarà a calcular el hash d'una petita quantitat de fitxers de mida gran. D'aquesta manera, el cost computacional de calcular un hash es molt elevat i les dades a comprimir i transmetre al següent host son poc voluminoses, minimitzant la interferència d'aquests factors.

A continuació s'exposa el temps en mil·lisegons que tarda el sistema a comprovar un determinat nombre de fitxers de 47MB en un host i informar al host ammaf:

| | 1 fitxer | 6 fitxers | 12 fitxers | 25 fitxers | 50 fitxers | 100 fitxers |
|---|---|---|---|---|---|---|
| mida targetsfile | 51B | 306B | 612B | 1.2KB | 2.5KB | 5KB |
| mida hashesfile | 116B | 0.7KB | 1.4KB | 2,8KB | 5.7KB | 11.3KB |
| Mostra 1 | 2371 | 11349 | 22216 | 45435 | 89563 | 178962 |
| Mostra 2 | 2431 | 11278 | 22058 | 45007 | 89503 | 179763 |
| Mostra 3 | 2386 | 11625 | 22182 | 45697 | 89100 | 178831 |
| Mostra 4 | 2467 | 11339 | 22116 | 45278 | 89540 | 178539 |

Figura 16: Estudi del rendiment al processar una gran quantitat de dades




Com es pot observar a la gràfica torna tractar-se d'un rendiment gairebé lineal, aspecte que confirma el fet de que les operacions de hash tenen un cost proporcional a la mida de les dades subministrades.

## 5.3 Estudi del rendiment de l'agent Signer

Segons els resultats experimentals, amb una quantitat petita de hosts, el temps mitjà que destina l'agent a generar una parella de claus a cada host i emmagatzemar-la al host principal es d'uns tres segons.

S'ha detectat que els valors d'aquest cost, per a un mateix nombre de hosts pot arribar a variar molt, havent-hi una diferència d'un 100% entre el valor mínim i el màxim.

## 5.4 Conclusions experimentals.

Una vegada vistos els rendiments en les diverses situacions extremes que s'han proposat se'n poden extreure conclusions i extrapolacions a altres configuracions de xarxa més grans.

Tenint en compte que la mida d'una clau publica i la seva signatura ocupa uns 550B i ja que es tracta de l'únic creixement que sofreix el paquet de claus que transporta l'agent Signer, la mida del paquet que ha de transportar no augmenta significativament

cosa que li proporciona un temps de empaquetament i transferència molt reduït. Això, juntament al fet que el cost en temps de la generació i certificació de parelles de claus ja es de per si reduït fan que aquest agent gaudeixi d'una escalabilitat excel·lent.

Com ja s'ha vist, el temps que triga el sistema a transportar fitxers de mida gran entre els hosts augmenta de forma lineal a la mida d'aquests arxius. Aquesta mida, per altra banda, augmenta a cada host visitat, ja que el fitxer de hashs que es genera sempre es mes gran que el fitxer llistat els fitxers a comprovar (un augment proporcional al nombre de fitxers a comprovar, sense que importi la seva mida). En aquest aspecte, el fet d'usar compressió de dades sacrifica temps de còmput a canvi de temps de transmissió. Això suposa un benefici, ja que canviar hosts i augmentar-ne la capacitat computacional es, parlant d'una xarxa gran de computadors d'usuari, més senzill que renovar tot l'equipament de xarxa.

Per altra banda, es pot observar que la velocitat de generació de hashs es sempre molt constant, ja que el volum de dades a comprovar no augmenta necessàriament al augmentar la quantitat de hosts a actualitzar, al contrari del que passa amb el volum de dades a transmetre.

Per tant, el cost en temps augmenta per cada host addicional que s'ha de verificar, ja que encara que els requisits de temps de càlcul per generar els hashs siguin sempre proporcionals al volum de dades que s'han de verificar, aquests hashs fan que la mida del paquet de dades a transferir augmenti i, per tant, el cost en temps per transferir-lo al següent host cada vegada és mes elevat. Tot i amb això, aquest augment es molt poc significatiu donat que el nombre de fitxers a verificar es molt improbable que superi els mil per host i per tant, el cost de transport dels fitxers es irrisori.

Finalment, en base a l'experimentació i anàlisis fets, es pot concloure que els requeriments de temps emprat en transmissió i compressió de dades son significativament menors que els necessaris per generar els fitxers de hash. Tot i amb això es tracta d'uns requeriments assumibles, ja que en una configuració del sistema típica, emprada per controlar fitxers crítics dels hosts d'usuari es molt improbable que els requisits arribin a ser comparables als valors mes baixos mostrats en aquest apartat d'experimentació, que ja de per si son valors assumibles.

# 6 Conclusions i treball futur

El disseny i implementació d'una eina enfocada a la seguretat no es gens trivial, i menys quan aquesta eina s'ha d'executar en un entorn distribuït i canviant. Tot i amb això en aquest projecte s'ha intentat amb bastant èxit crear un prototip amb les funcionalitats bàsiques d'un sistema d'aquestes característiques. El prototip necessita ésser refinat, i els seus paràmetres de funcionament han d'ésser estudiats mes minuciosament abans no es pugui implementar en una xarxa de producció, però tot i amb això, es tracta d'un prototip plenament funcional.

Per altra banda, l'experimentació realitzada demostra que el sistema en té un rendiment acceptable tenint en compte la quantitat d'operacions de càlcul i criptogràfiques que empra per minimitzar la intrusió en l'equipament de xarxa i mantenir una gran resistència a atacs provinents d'usuaris que intentin comprometre la xarxa o extreure'n informació utilitzant el sistema.

Com a aspecte negatiu, mencionar que el sistema hereta de JADE el problema de que aquest es poc segur, cosa que augmenta el risc de sofrir atacs d'agents mòbils malformats. Aquest risc ve motivat pel fet que la incorporació d'un contenidor a una plataforma no és verificat per el contenidor principal, de manera que qualsevol pot afegir un contenidor a una plataforma existent. Afortunadament el desenvolupament de l'afegit JADE-S avança ràpidament i subsanarà aquest problema, així com facilitarà la implementació d'esquemes de seguretat i autenticació.

Per altra banda, queden com a possibles millores del sistema:

- Implementar un agent addicional que comprovi els hashs una vegada aquests han estat generats, enlloc de realitzar aquesta comprovació de forma manual al host ammaf una vegada s'obtenen nous hashs.
- Coordinar una major quantitat d'agents simultàniament per augmentar la robustesa del sistema i reduir la sobrecàrrega que suposa que una mateixa plataforma cobreixi una gran quantitat de hosts.
- Dotar als agents d'un llenguatge de comunicació que els permeti rebre ordres i/o actuar amb més autonomia segons el seu estat i condicions d'execució.
- Depurar el codi font actual i mantenir-lo operatiu amb noves versions de JADE i Java.
- Dissenyar i implementar un sistema de registre d'activitats dels agents per tal de facilitar la detecció de problemes.
- Modificar el sistema d'agents per emprar JADE-S quan aquest sigui capaç de garantir la seguretat i autenticitat de totes les plataformes i a més a més sigui operatiu amb plataformes que permetin la mobilitat d'agents.

# 7 Agraiments

Per acabar, voldria donar les gràcies al meu tutor, en Francesc Solsona, per ajudar-me amb el projecte. Així com a la meva família que m'han donat ànims per poder acabar el projecte i m'han suportat quan les coses no han anat bé. A tots ells, gràcies.

# A Codis font del sistema d'agents

A continuació es mostra el codi font de totes les classes java implementades per a aquest projecte.

## A.1 Signer.java

```
package ammaf;
import jade.core.*;
import jade.core.behaviours.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
//mobility
import jade.content.*;
import jade.content.onto.basic.*;
import jade.content.lang.Codec;
import jade.content.lang.sl.SLCodec;
import jade.domain.mobility.MobilityOntology;
import jade.domain.JADEAgentManagement.*;
import jade.lang.acl.*;
import jade.proto.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import ammaf.util.*;
import java.io.*;
/**This agent generates and signs the keypairs of all the to be checked hosts and returns the home host
 * to store its public key and signature.
 * Note that this agent needs a target hosts file named "hosts.list" to know where should it move to.
 * The "hosts.list" file should consist in a simple text file that contains host names separated by carry returns.
 *
@author Gatsu
@version 0.1
*/
public class Signer extends Agent {

 /** Some widely used values */
 private String hostsfilename = "hosts.list";
 private String zipfilename = "signingpackage.zip";
 private String keysdir = "keys";

 private String pubkeyfilename = "pubkey.key";
 private String pubkeysignaturefilename = "pubkey.sig";
 private String prikeyfilename = "prikey.key";
 private String prikeysignaturefilename = "prikey.sig";
 private String signingpubkeyfilename = "master_pubkey.key";
 private String signingprikeyfilename = "master_prikey.key";
 private String agent_type = "Signer type"; //The type that will be registered in the DF, an arbitrary name
 private String senddata = "Send me the data please";
 /** Key variables */
 protected KeyPair signingkeypair;

 protected transient byte[] hpublic;
 protected transient byte[] hprivate;
 protected transient byte[] pubksignature;
 protected transient byte[] priksignature;
 /** Other variables */
 private boolean cloned=false; //defines if the agent has been cloned in the current host
 private boolean original=false; //defines if the agent is the original one
 protected boolean error=false; //defines if there was any error
 private boolean returnhome=false; //defines if should return home
 protected boolean checkcurrent=false;
 protected String checkhostname;
 protected String homename;
```

```
private transient Location checkhostl;
/**Initialization method*/
protected void setup(){
 //this is the original agent
 original=true;

 //save the home adress
 homename= here().getAddress();

  // Register language and ontology
 getContentManager().registerLanguage(new SLCodec());
 getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
 getContentManager().registerOntology(MobilityOntology.getInstance());
 //should register with the df, not necesary but could be used in the future in order to add new features
 try {
  System.out.println(getLocalName()+": Registering with the DF service");
  DFAgentDescription dfad = new DFAgentDescription();
  dfad.setName(getAID());
  ServiceDescription sd = new ServiceDescription();
  sd.setType(agent_type);
  sd.setName(getName());
  sd.setOwnership("UDL");
  dfad.addServices(sd);
  DFService.register(this, dfad);
 } catch (Exception e) {
  System.err.println(getLocalName()+": Registration with the DF service unsucceeded. ABORTING EXECUTION");
  error=true;
  doDelete();
 }
 /// Adds the required behaviours (It is not really necessary to do this,but it eases
 //possible modifications like storing the state of the agent in a file or adding other
 //complex functionalityes)
 addBehaviour( new PrepareSigningKeypair (this) );
 //FIXME:note that this behaviour adds more behaviours depending on the host, perhaps would be more correct to
 //use a FSM behaviour, especially if this agent has to be expanded with new functionalityes
}
/** A behaviour that prepares (loads or generates) the master keypair if the hosts file contains at least one host.
 * This behaviour also queues the necesary behaviours to perform the sign operation in the current host*/
class PrepareSigningKeypair extends OneShotBehaviour{
 public PrepareSigningKeypair(Agent a){
  super(a);
 }
 public void action(){
  try{
   BufferedReader hosts_buffer = new BufferedReader(new FileReader(hostsfilename));

   /** Checks if there is any host, there is no need to generate a keypair if there aren't hosts */
   if( (checkhostname = hosts_buffer.readLine()) != null){
    hosts_buffer.close();
    System.out.println(getLocalName()+": Looking for a previously generated signing keypair");
    File spubkeyf = new File(signingpubkeyfilename);
    File sprikeyf = new File(signingprikeyfilename);
    if (spubkeyf.canRead() && sprikeyf.canRead() ){

     /** Loads the signing keypair */
     System.out.println(getLocalName()+": A signing keypair was found, loading it");
     FileInputStream pubsignis = new FileInputStream(signingpubkeyfilename);
     FileInputStream prisignis = new FileInputStream(signingprikeyfilename);
     if (pubsignis.available()>2048 || prisignis.available()>2048){
      System.err.println(getLocalName()+": The signing keys' files are too big. ABORTING EXECUTION");
      //should abort the whole operation
      error=true;
      myAgent.doDelete();
     }
     else{
      byte[] spubkey = new byte[pubsignis.available()];
      byte[] sprikey = new byte[prisignis.available()];
```

```
      pubsignis.read( spubkey ); //Warning! reads the entire file (it should be a small file)
      prisignis.read( sprikey ); //checked before.
      X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec( spubkey );
      PKCS8EncodedKeySpec priKeySpec = new PKCS8EncodedKeySpec( sprikey );
      KeyFactory keyfactory = KeyFactory.getInstance("RSA");
      signingkeypair = new KeyPair ( keyfactory.generatePublic(pubKeySpec),
                                     keyfactory.generatePrivate(priKeySpec) );
      pubsignis.close();
      prisignis.close();
     }
    }
    else {
     /** Generates a new signing keypair */
     System.out.println(getLocalName()+
         ": A signing keypair could NOT be found\n   Generating a new signing keypair, please wait...");
     KeyPairGenerator keygen = KeyPairGenerator.getInstance( "RSA" );
     keygen.initialize( 2048 ); //key size
     signingkeypair = keygen.generateKeyPair();

     System.out.println(getLocalName()+": Storing signing keypair at home");
     /**Stores the signing keypair*/
     FileOutputStream pubsignos = new FileOutputStream(signingpubkeyfilename);
     FileOutputStream prisignos = new FileOutputStream(signingprikeyfilename);
     pubsignos.write( signingkeypair.getPublic().getEncoded() );
     prisignos.write( signingkeypair.getPrivate().getEncoded() );
     pubsignos.close();
     prisignos.close();
    }
   }
   else{
    System.err.println(getLocalName()+": "+ hostsfilename+" was not found. ABORTING EXECUTION");
    error=true;
    myAgent.doDelete();
   }
   checkcurrent= homename.equals(checkhostname);
   if(checkcurrent){
    addBehaviour( new GenerateHostKeypair (myAgent) );
    addBehaviour( new SignHostKeypair (myAgent) );
    addBehaviour( new StoreKeysInHost (myAgent) );
    addBehaviour( new PackageFiles (myAgent, true) );
   }
   else{
    addBehaviour( new PackageFiles (myAgent, false) );
   }
   addBehaviour( new AskForMigration(myAgent) );
  }catch(Exception e){
   System.err.println(getLocalName()+": Error preparing the master keypair");
   error=true;
   myAgent.doDelete();
  }
 }
}
/** This behaviour generates the host keypair.*/
class GenerateHostKeypair extends OneShotBehaviour{
 public GenerateHostKeypair(Agent a){
  super(a);
 }
 public void action(){
  try{
   System.out.println(getLocalName()+": Generating host keypair");
   KeyPairGenerator hkeygen = KeyPairGenerator.getInstance( "RSA" );
   hkeygen.initialize( 2048 );
   KeyPair hostkeypair = hkeygen.generateKeyPair();
   hpublic = hostkeypair.getPublic().getEncoded();
   hprivate = hostkeypair.getPrivate().getEncoded();
  }catch(Exception e){
   System.err.println(getLocalName()+": Error generating the host keypair. ABORTING EXECUTION");
   error=true;
```

```
   myAgent.doDelete();
  }
 }
}
/** This behaviour generates the host keypair signature.*/
class SignHostKeypair extends OneShotBehaviour{
 public SignHostKeypair(Agent a){
  super(a);
 }
 public void action(){
  try{
   System.out.println(getLocalName()+": Signing the host keypair");
   Signature rsa = Signature.getInstance( "SHA1withRSA" );
   rsa.initSign( signingkeypair.getPrivate() );
   rsa.update( hpublic );
   pubksignature = rsa.sign();
   rsa.update( hprivate );
   priksignature = rsa.sign();
  }catch(Exception e){
   System.err.println(getLocalName()+": Error signing the host keypair. ABORTING EXECUTION");
   error=true;
   myAgent.doDelete();
  }
 }
}
/** This behaviour stores the keypair and its signature at the host.*/
class StoreKeysInHost extends OneShotBehaviour{
 public StoreKeysInHost(Agent a){
  super(a);
 }
 public void action(){
  try{
   System.out.println(getLocalName()+": Storing the host's keys");
   FileOutputStream storeos = new FileOutputStream(pubkeyfilename);
   storeos.write( hpublic );
   storeos.close();

   storeos = new FileOutputStream(pubkeysignaturefilename);
   storeos.write( pubksignature );
   storeos.close();

   storeos = new FileOutputStream(prikeyfilename);
   storeos.write( hprivate );
   storeos.close();

   storeos = new FileOutputStream(prikeysignaturefilename);
   storeos.write( priksignature );
   storeos.close();
   //Unreference all the keys to avoid serializing and moving them to the next host,
   //also suggest to pass the garbage collector
   hpublic = null;
   pubksignature = null;
   hprivate = null;
   priksignature = null;
   java.lang.System.gc();
  }catch(Exception e){
   System.err.println(getLocalName()+": Error storing the host's keys. ABORTING EXECUTION");
   error=true;
   myAgent.doDelete();
  }
 }
}
/** This behaviour creates the to be transfered package. The package contains the files of the previously
 * created package and if the boolean packagekey is true, it also packages the host public key and its
 * signature files */
class PackageFiles extends OneShotBehaviour{
 boolean packagekey;
 public PackageFiles(Agent a, boolean packagekey){
```

```
 super(a);
 this.packagekey=packagekey;
}
public void action(){
 //Copy the public key
 //compress and add the public key
 try{
  System.out.println(getLocalName()+": Packaging the host public key and its signature");
  File zipfile = new File (zipfilename);
  boolean exists = zipfile.canRead();
  PackagingTools pt;
  //The zip file does not exist, should add the hosts file
  if (exists){
   pt = new PackagingTools(zipfilename, "add files");
  }
  else{
   pt = new PackagingTools(zipfilename, "new package");
   pt.addToPackage(hostsfilename, hostsfilename);
  }
  if (packagekey){
   //adds the pulbic key file
   pt.addToPackage(pubkeyfilename, myAgent.here().getAddress() + ".key" );
   //adds the current host key signature file to the pack
   pt.addToPackage(pubkeysignaturefilename, myAgent.here().getAddress() + ".sig" );
   //should be getName() and use container names instead of host names if
   //we wish to allow several containers in each host
  }

  //finally finishes the package
  pt.finishPackaging();
 }catch ( Exception e){
  System.err.println(getLocalName()+": Error packaging the key and its signature. ABORTING EXECUTION");
  error=true;
  myAgent.doDelete();
 }
}
}
/** This behaviour querries the ams for possible destinations and searches the destination found in*/
class AskForMigration extends OneShotBehaviour{
 public AskForMigration(Agent a){
  super(a);
 }
 public void action(){
  try{
   boolean found=false;
   //Queryes the ams in order to get the possible destinations
   //Fills the message fields
   ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
   msg.addReceiver(getAMS());
   msg.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
   msg.setOntology(MobilityOntology.NAME);
   msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
   Action action = new Action();
   action.setActor(getAMS());
   action.setAction(new QueryPlatformLocationsAction());
   getContentManager().fillContent(msg, action);
   send(msg);
   //Receive response from AMS
   MessageTemplate mt = MessageTemplate.and(
     MessageTemplate.MatchSender(getAMS()),
     MessageTemplate.MatchPerformative(ACLMessage.INFORM));
   mt = MessageTemplate.and( mt, MessageTemplate.MatchOntology(MobilityOntology.NAME));
   mt = MessageTemplate.and( mt, MessageTemplate.MatchProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST));
   mt = MessageTemplate.and( mt, MessageTemplate.MatchLanguage(FIPANames.ContentLanguage.FIPA_SL0) );
   msg = blockingReceive(mt); //Warning, if the ams does not respond the execution of the agent stops forever
   Result result = (Result)getContentManager().extractContent(msg);
   jade.util.leap.Iterator it = result.getItems().iterator();
```

```
   if(checkcurrent){
    BufferedReader hosts_buffer = new BufferedReader(new FileReader(hostsfilename));
    //Checks if there is another host
    String readhostname=null;
    do{
     readhostname = hosts_buffer.readLine();
    }while(!checkhostname.equals(readhostname) && readhostname!=null);
    checkhostname = hosts_buffer.readLine();
    hosts_buffer.close();
   }
   checkcurrent=true;
   if(checkhostname == null){
    checkhostname = new String(homename);
    returnhome=true;
   }
   //if already at home and it's the last host, there is no need to move
   if (returnhome && homename.equals(myAgent.here().getAddress())){
    myAgent.doDelete();
   }
   else{
    //tryes to match the location with the next one in the hosts file
    checkhostl=null;
    System.out.println(getLocalName()+": Searching for locations.\n   Found locations:");
    while (it.hasNext() && !found) {
     checkhostl = (Location)it.next();
     System.out.println("      "+checkhostl.getAddress());
     found = checkhostl.getAddress().equals(checkhostname);
    }
    if (!found){
     System.err.println( getLocalName()+": Could not find the host "+checkhostname+". ABORTING EXECUTION");
     error=true;
     myAgent.doDelete();
    }
    else{
     System.out.println(getLocalName()+": Being cloned");
     //Before clonning, we add a behaviour that blocks until the clone operation
     //is finished in order to migrate
     addBehaviour(new Migration(myAgent));
     doClone(myAgent.here(), getLocalName()+" helper");
    }
   }
  }catch(Exception e){
   System.err.println(getLocalName()+": Error requesting a migration to the next host. ABORTING EXECUTION");
   error=true;
   myAgent.doDelete();
  }
 }
}
/** A simple behaviour that allows the agent to move to next host when cloned*/
class Migration extends SimpleBehaviour{
 public Migration(Agent a){
  super(a);
 }
 public void action(){
  //checks if the agent has been cloned
  if (cloned && original){
   myAgent.doMove(checkhostl);
  }
  else{
   //blocks the behaviour awaiting for an event
   block();
  }
 }
 public boolean done(){
  //finishes the behaviour if the agent has been cloned, if not, it is rescheduled automatically
  //until a new event unblocks the behaviour
  return (cloned);
 }
```

```
}
/** An AchieveREInitiator behaviour that handles the file transfer at the original*/
class TransferRequest extends AchieveREInitiator{
 FileOutputStream filefos;
 StoreIncomingFile sifb;
 public TransferRequest(Agent a, ACLMessage request){
  super(a, request);
 }
 //When the helper agrees to transmit the file, the original should create the file
 //in the current host
 protected void handleAgree(ACLMessage agree){
  try{
   filefos= new FileOutputStream(zipfilename);
   //must prepare for the incoming messages that contain the file
   //add StoreIncomingFile (it is saved in order to remove it when finished)
   sifb = new StoreIncomingFile(myAgent, filefos);
   myAgent.addBehaviour(sifb);
  }catch(Exception e){
   System.err.println(getLocalName()+": Error recieving file");
   error=true;
   doDelete();
  }
 }
 protected void handleInform(ACLMessage inform) {
  try{
   //filefos.close();
   Hash hash = new Hash();
   if(hash.hashOfFile(zipfilename).equals(inform.getContent())){
    System.out.println(getLocalName()+ ": File transfer completed succesfully, unpacking files");
    //unpacks the hosts file
    if(!returnhome){
     PackagingTools pt = new PackagingTools(zipfilename, "unpack");
     pt.unpackFileTo(".", hostsfilename);
     pt.finishUnpacking();
    }
   }
   else{
    System.err.println(getLocalName()+": File's hashes mismach");
    error=true;
    doDelete();
   }
   //finished the transmission, must remove StoreIncomingFile
   myAgent.removeBehaviour(sifb);
  }catch(Exception e){
   error=true;
   doDelete();
  }
 }
 protected void handleOutOfSequence(ACLMessage msg){
  System.err.println(getLocalName()+": Message out of sequence");
  error=true;
  doDelete();
 }
 protected void handleRefuse(ACLMessage refuse){
  System.err.println(getLocalName()+": The helper refused to send the file");
  error=true;
  doDelete();
 }
 //The behaviour is already removed, so we can migrate
 public int onEnd(){
  if(!returnhome){
   addBehaviour( new GenerateHostKeypair (myAgent) );
   addBehaviour( new SignHostKeypair (myAgent) );
   addBehaviour( new StoreKeysInHost (myAgent) );
   addBehaviour( new PackageFiles (myAgent, true) );
   addBehaviour( new AskForMigration(myAgent) );
  }
  else{
```

```
   try{
    System.out.println("\nNETWORK KEYPAIR GENERATION SUCCESFULLY FINISHED\n");
    //Decompress the package to the keys directory
    PackagingTools pt = new PackagingTools(zipfilename, "unpack");
    File f = new File (keysdir);
    if (!f.exists()){
     f.mkdirs();
    }
    pt.unpackFileTo(".", hostsfilename);
    if (f.isDirectory()){
     pt.unpackTo(keysdir);
     f = new File(keysdir + File.separator + hostsfilename);
     f.delete();
    }
    pt.finishUnpacking();
    f = new File(zipfilename);
    f.delete();
   }catch(Exception e){
    error=true;
    System.err.println(getLocalName()+": WARNING, there was an error unpacking the files");
   }finally{
    doDelete();
   }
  }
  return 0;
 }
}
/** An AchieveREResponder behaviour that handles the file transfer at the helper agent. When the file transfer is
 * finished, it queues a delete behaviour*/
class TransferReply extends AchieveREResponder{
 public TransferReply(Agent a, MessageTemplate mt){
  super(a, mt);
 }
 protected ACLMessage prepareResponse(ACLMessage request){
  ACLMessage response = request.createReply();
  response.setPerformative(ACLMessage.AGREE);
  return response;
 }
 protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage response) {
  ACLMessage informDone=null;
  try{
   FileInputStream fis = new FileInputStream(zipfilename);
   //send the file
   ACLMessage filemessage= new ACLMessage(ACLMessage.UNKNOWN);
   filemessage.addReceiver(request.getSender());
   byte[] filechunk = new byte[8192];//8kb
   byte[] sendchunk;
   int readbytes;
   while ( ( readbytes = fis.read( filechunk )) != -1){
    sendchunk= new byte[readbytes];
    for ( ; readbytes !=0; readbytes--){
     sendchunk[readbytes-1]=filechunk[readbytes-1];
    }
    filemessage.setContentObject(sendchunk);
    send(filemessage);
   }
   //inform and finish the protocol
   informDone = request.createReply();
   informDone.setPerformative(ACLMessage.INFORM);
   Hash hash = new Hash();
   //sends the file hash in order to check it SHOULD CHANGE IT BECAUSE THE FILE IS READ 2 TIMES
   informDone.setContent(hash.hashOfFile(zipfilename));
   //deletes the zipfile in this host
   File zipfile = new File(zipfilename);
   zipfile.delete();
   //everything is finished scheduling a simple suicidal behaviour
   addBehaviour( new OneShotBehaviour(myAgent){
    public void action(){
```

```
     myAgent.doDelete();
    }
   });
  }catch(Exception e){
   System.err.println(getLocalName()+": Error sending file");
   error=true;
   doDelete();
  }
  return informDone;
 }
}
/** Recieves and stores the remote file.*/
class StoreIncomingFile extends SimpleBehaviour{
 MessageTemplate mt;
 ACLMessage msg;
 FileOutputStream filefos;
 byte[] filebytes;
 public StoreIncomingFile(Agent a, FileOutputStream filefos){
  super(a);
  mt = MessageTemplate.and(
    MessageTemplate.MatchSender(new AID(getLocalName()+" helper", AID.ISLOCALNAME)),
    MessageTemplate.MatchPerformative(ACLMessage.UNKNOWN));
  this.filefos=filefos;
 }
 public void action(){
  msg = myAgent.receive(mt);
  if (msg != null) {
   try{
    //A message that matches the template has been recieved. Process it
    //note that this method isn't fipa compilant, but it is a relatively efficient way
    //of transmitting files (not too much overheat and the messages still have correct
    //fipa headings
    filebytes = (byte[])msg.getContentObject();
    filefos.write(filebytes);
   }catch(Exception e){
    System.err.println(getLocalName()+": Error recieving file");
    error=true;
    doDelete();
   }
  }
  //blocks the behaviour awaiting another message
  block();
 }
 public boolean done(){
  return false;
 }
 public int onEnd(){
  try{
   filefos.close();
  }catch(Exception e){
   System.err.println(getLocalName()+": Error recieving file");
   error=true;
   doDelete();
  }
  return 1;
 }
}
protected void beforeMove() {
}
protected void afterMove() {
 cloned=false;
 System.out.println(getLocalName()+": I have reached your host");
 // Register language and ontology
 getContentManager().registerLanguage(new SLCodec());
 getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
 getContentManager().registerOntology(MobilityOntology.getInstance());
 //addBehaviour(new RequestDataBehaviour());
```

```
  /** Requests the file transfer to the clone*/
  ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
  request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
  request.addReceiver(new AID(getLocalName()+" helper", AID.ISLOCALNAME));
  request.setContent(senddata);
  addBehaviour(new TransferRequest(this, request));
 }
 /** Sets the cloned variable to true. Triggered in both the agent and the clone before cloning*/
 protected void beforeClone() {
  try{
   cloned=true;
  }catch(Exception e){
   System.err.println(getLocalName()+": Error migrating. ABORTING EXECUTION");
  }
 }
 /** Deletes the hosts file and prepares for the incoming transfer adding a TransferReplyBehaviour. Triggered
  * in the clone when finished cloning*/
 protected void afterClone() {
  try{
   original=false;
   File hostsfile = new File (hostsfilename);
   hostsfile.delete();
   //waits for an "arrived" message
   MessageTemplate mt = AchieveREResponder.createMessageTemplate(FIPANames.InteractionProtocol.FIPA_REQUEST);
   mt = MessageTemplate.and( mt, MessageTemplate.MatchContent(senddata));

   this.addBehaviour( new TransferReply(this, mt));
  }catch(Exception e){
   System.err.println(getLocalName()+": Error sending the files");
   error=true;
   doDelete();
  }
 }
 /** Deregisters the agent entry in the df (if the agent is not a helper).*/
 protected void takeDown(){
  try{
   if(!cloned){
    //Deregisters its entry in the df (only the original agent registers with the DF service
    //because the clone is just a helper for the file transfers.
    DFService.deregister(this);
   }
   if (!error){
    System.out.println(getLocalName()+": I have finished my job");
   }
   else{
    System.err.println(getLocalName()+": Finishing due to an error");
   }
  }catch (Exception e){
   System.err.println("Error destroying the agent");
  }
 }
}
```

## A.2 Decrypter.java

```
package ammaf;
import jade.core.*;
import jade.core.behaviours.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.proto.*;
import jade.lang.acl.*;
import jade.wrapper.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
```

```
import java.io.*;
import java.lang.reflect.Array;
import ammaf.util.*;
/**This agent starts the system and when the whole process has been completed, it decrypts
 * and reports the results. Needs the following things:
 * The parameter update to start the update process (Done to structure and ease future
 * modifications to the system).
 * A signed keypair in every host, a master keypair in the current directory and the public
 * keys and its signatures in the
 * subdirectory keys (so first the agent system should run the Signer agent).
 * A updatehosts.list file that contains the hostnames of the to be checked host separated
 * by carry returns.
 * A subdirectory called targetsdir that contains a targets file for each host separated by
 * carry returns that needs to be named [hostname].targets
@author Gatsu
@version 0.1
*/
public class Decrypter extends Agent {
 protected Key privatekey;
 private String algorithm = "SHA-256";
 private String pubkeyfilename = "pubkey.key";
 private String pubkeysfilename = "pubkey.sig";
 private String prikeyfilename = "prikey.key";
 private String prikeysfilename = "prikey.sig";
 private String signingpubkeyfilename = "master_pubkey.key";
 private String signingprikeyfilename = "master_prikey.key";
 protected boolean error=false;
 private String keysdir = "keys";
 private String targetsdir = "targetfiles";
 private String hashesdir = "hashes";

 private String updatepackagename = "updatepackage.zip";
 private String updatetargethostsfilename = "updatehosts.list";
 private String updatefinishmessage = "The update process has finished succesfully";
 protected String mode;
 protected SecretKey secretkey;

 /**Initialization method*/
 protected void setup(){
  //should register with the df, not necesary but could be used in the future in order to add new features
  try {
   Object[] arguments = getArguments();
   if ( Array.getLength(arguments) != 1 ){
    error = true;
    System.err.println(getLocalName()+": Incorrect number of arguments");
    doDelete();
   }
   mode = (String)arguments[0];
   System.out.println(getLocalName()+": Registering in the DF service");
   DFAgentDescription dfad = new DFAgentDescription();
   dfad.setName(getAID());
   ServiceDescription sd = new ServiceDescription();
   sd.setType("Decrypter");
   sd.setName(getName());
   sd.setOwnership("UDL");
   dfad.addServices(sd);
   DFService.register(this, dfad);
  } catch (Exception e) {
   System.err.println(getLocalName()+": Registration with the DF service unsucceeded. ABORTING EXECUTION");
   error=true;
   doDelete();
  }
  //Adds the required behaviours (It is not really necessary to do this,but it eases
  //possible modifications like storing the state of the agent in a file or adding other
  //complex functionalityes)
  addBehaviour ( new Scheduler(this));
 }
 /** Controls and schedules all behaviours*/
```

```
class Scheduler extends FSMBehaviour{
 Decrypter d;
 public Scheduler (Agent a){
  super(a);
  d=(Decrypter)a;
  //Register state behaviours
  if(d.mode.equals("update")){
   registerFirstState(new PackageUpdaterFiles(a), "packaging updater files");
   registerState(new StartUpdater(a), "starting updater");
   MessageTemplate mt = AchieveREResponder.createMessageTemplate(FIPANames.InteractionProtocol.FIPA_REQUEST);
   mt = MessageTemplate.and( mt, MessageTemplate.MatchSender(new AID("Updater", AID.ISLOCALNAME)));
   registerState(new ProcessUpdate(a, mt), "processing update");
   registerLastState(new ReportUpdateResults(a), "reporting");
   //Transitions
   registerDefaultTransition("packaging updater files", "starting updater");
   registerDefaultTransition("starting updater", "processing update");
   registerDefaultTransition("processing update", "reporting");
  }
  else {
   d.error=true;
   d.doDelete();
  }
  scheduleFirst();
 }
}
/** Gathers all the needed files and creates a package file. Uses zip compresion
 * provided by the PackagingTools class */
class PackageUpdaterFiles extends OneShotBehaviour{
 public PackageUpdaterFiles (Agent a){
  super (a);
 }
 public void action(){
  try{
   System.out.println(getLocalName()+": Packaging the files needed to perform an update");
   File kdf = new File(keysdir);
   File tdf = new File(targetsdir);
   String targethost;
   if (kdf.isDirectory() && tdf.isDirectory()){
    PackagingTools pt = new PackagingTools(updatepackagename, "new package");
    pt.addToPackage(updatetargethostsfilename, updatetargethostsfilename);
    BufferedReader hosts_buffer = new BufferedReader(new FileReader(updatetargethostsfilename));
    //adds the home plataform keys
    pt.addToPackage(keysdir + File.separator + myAgent.here().getAddress() +
                    ".key", myAgent.here().getAddress() + ".key");
    pt.addToPackage(keysdir + File.separator + myAgent.here().getAddress() +
                    ".sig", myAgent.here().getAddress() + ".sig");

    while ((targethost = hosts_buffer.readLine()) != null){
     pt.addToPackage(keysdir + File.separator + targethost + ".key", targethost + ".key");
     pt.addToPackage(keysdir + File.separator + targethost + ".sig", targethost + ".sig");
     pt.addToPackage(targetsdir + File.separator + targethost + ".targets", targethost + ".targets");
    }
    pt.finishPackaging();
    hosts_buffer.close();
    File updatehostsfile = new File (updatetargethostsfilename);
    updatehostsfile.delete();
   }
  }catch(Exception e){
   error=true;
   System.err.println(getLocalName()+": Error packaging the required files to perform an update");
   myAgent.doDelete();
  }
 }
}
/** Prepares the Decryption keypair and passes its public part to the updater.*/
class StartUpdater extends OneShotBehaviour{
 public StartUpdater (Agent a){
  super(a);
```

```
}
public void action(){
 try{
  PublicKey signerpubkey=null;
  //Start the checker agent at the current host
  PlatformController pc = getContainerController();
  Object[] wrapper = new Object[3];
  wrapper[0]=algorithm;

  File spubkeyf = new File(signingpubkeyfilename);
  File sprikeyf = new File(signingprikeyfilename);
  if (spubkeyf.canRead() && sprikeyf.canRead() ){
   // Loads the signing public key
   FileInputStream pubsignis = new FileInputStream(signingpubkeyfilename);
   if (pubsignis.available()>2048){
    System.err.println(getLocalName()+": The signing keys' files are too big. ABORTING EXECUTION");
    error=true;
    myAgent.doDelete();
   }
   else{
    byte[] spubkey = new byte[pubsignis.available()];
    pubsignis.read( spubkey ); //Warning! reads the entire file (it should be a small file)
    X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec( spubkey );
    KeyFactory keyfactory = KeyFactory.getInstance("RSA");
    signerpubkey = keyfactory.generatePublic(pubKeySpec);
    wrapper[1] = signerpubkey;
    pubsignis.close();
   }
  }
  else{
   System.err.println(getLocalName()+": Target and/or key directories are missing");
   error=true;
   myAgent.doDelete();
  }
  // Loads the main host public key
  FileInputStream pubkeyis = new FileInputStream(pubkeyfilename);
  FileInputStream pubkeysignis = new FileInputStream(pubkeysfilename);
  if (pubkeyis.available()>2048 || pubkeysignis.available()>2048){
    System.err.println(getLocalName()+": The main host public key files are too big. ABORTING EXECUTION");
    error=true;
    myAgent.doDelete();
  }
  else{
   byte[] spubkey = new byte[pubkeyis.available()];
   byte[] pubkeysignature = new byte[pubkeysignis.available()];

   pubkeyis.read( spubkey );
   pubkeysignis.read( pubkeysignature );
   //Warning! reads the entire file (it should be a small file)
   //checked before.

   //Checks if the private key is signed by the signer authority
   Signature rsasignature = Signature.getInstance("SHA1withRSA");
   rsasignature.initVerify(signerpubkey);
   rsasignature.update(spubkey);
   if( rsasignature.verify(pubkeysignature)){
    //The key is correctly signed, parse it
    X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec( spubkey );
    KeyFactory keyfactory = KeyFactory.getInstance("RSA");
    PublicKey mainhostpubkey = keyfactory.generatePublic(pubKeySpec);
    //generate an aes key and encrypts it with the main host public key
    KeyGenerator keygen = KeyGenerator.getInstance("AES");
    secretkey = keygen.generateKey();
    Cipher c1 = Cipher.getInstance("RSA");
    c1.init(Cipher.WRAP_MODE, mainhostpubkey);
    wrapper[2]=(c1.wrap(secretkey));
    System.out.println(getLocalName()+": Starting the Updater agent");
    pc.createNewAgent("Updater", "udlfam.Updater", wrapper).start();
```

```
    }
   }
  }catch (Exception e){
   error=true;
   System.err.println(getLocalName()+": Error Starting the Updater agent");
   myAgent.doDelete();
  }
 }
}
/** Awaits for the completion of the update process and attends the request to store the results*/
class ProcessUpdate extends AchieveREResponder{
 public ProcessUpdate(Agent a, MessageTemplate mt){
  super(a, mt);
 }
 protected ACLMessage prepareResponse(ACLMessage request){
  System.out.println(myAgent.getLocalName()+": Checking if the sender is really the updater");
  ACLMessage response = null;
  try{
   Cipher c1 = Cipher.getInstance("AES");
   c1.init(Cipher.DECRYPT_MODE, secretkey);
   String cleartext = new String (c1.doFinal((byte[])request.getContentObject()));
   if (!cleartext.equals(updatefinishmessage)){
    response = request.createReply();
    response.setPerformative(ACLMessage.REFUSE);
    //probably will not be correctly decrypted, because the updater used an incorrect
    //secret key, but can be used in order to allow the detection of fake refuse
    //messages or fake finish messages
    c1.init(Cipher.ENCRYPT_MODE, secretkey);
    response.setContentObject(c1.doFinal("REFUSE".getBytes()));
   }
  }catch(Exception e){
   error=true;
   System.err.println(getLocalName()+": Exception processing the Updater request");
   e.printStackTrace();
   myAgent.doDelete();
  }
  //can be null (no response is sent and the updater just recieves the informative notification,
  //it is a correct behaviour that follows the FIPA standars);
  //or a refuse message if the decrypted request message does not match the expected one
  return response;
 }
 //called when the request message was correctly encrypted
 protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage response) {
  //sends the secret key encrypted using the next host public key
  try{
   Cipher c1 = Cipher.getInstance("AES");
   c1.init(Cipher.ENCRYPT_MODE, secretkey);
   response = request.createReply();
   response.setPerformative(ACLMessage.INFORM);
   response.setContentObject(c1.doFinal("AGREED TO CHECK THE FILES".getBytes()));
  }catch(Exception e){
   error=true;
   System.err.println(getLocalName()+": Error sending the inform message");
   e.printStackTrace();
   myAgent.doDelete();
  }
  //Adds the report behaviour
  myAgent.addBehaviour(new ReportUpdateResults(myAgent));
  return response;
 }
}
/** Unpacks the generated hashes files */
class ReportUpdateResults extends OneShotBehaviour{
 public ReportUpdateResults (Agent a){
  super (a);
 }
 public void action(){
  try{
```

```
    //Decompress the hashes to the hashes directory
    File f = new File (hashesdir);
    //Create the hashes directory if it does not exist
    if (!f.exists()){
     f.mkdirs();
    }
    PackagingTools pt = new PackagingTools(updatepackagename, "unpack");
    BufferedReader hosts_buffer = new BufferedReader(new FileReader(updatetargethostsfilename));
    String readhostname;
    do{
     readhostname = hosts_buffer.readLine();
     if(readhostname !=null){
      if (pt.hasFile(readhostname+".hashes")){
       pt.unpackFileTo(hashesdir, readhostname+".hashes");
      }
      else{
       System.out.println(getLocalName()+": WARNING!, the "+readhostname+" hashes could not be found,
                          probably that host was not online");
      }
     }
    }while(readhostname!=null);
    pt.finishUnpacking();
    f = new File(updatepackagename);
    f.delete();
    System.out.println("\nHASHES UPDATE CORRECTLY FINISHED\n");
   }catch(Exception e){
    error=true;
    System.err.println(getLocalName()+": WARNING, there was an error unpacking the files, most likely
                       some hosts could not be visited and they hashes files are not available\nKeeping
                       the update package for manual examination");
   }finally{
    myAgent.doDelete();
   }
  }
 }
 /** This method takes down the agent*/
 protected void takeDown(){
  try{
   DFService.deregister(this);
   System.out.println(getLocalName()+": Terminating");
  }catch (Exception e){
   System.err.println("Error destroying the agent");
  }
 }
}
```

## A.3 Updater.java

```
package ammaf;
import jade.core.*;
import jade.core.behaviours.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
//mobility
import jade.content.*;
import jade.content.onto.basic.*;
import jade.content.lang.Codec;
import jade.content.lang.sl.SLCodec;
import jade.domain.mobility.MobilityOntology;
import jade.domain.JADEAgentManagement.*;
import jade.lang.acl.*;
import jade.proto.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import ammaf.util.*;
```

```
import java.io.*;
import java.util.*;
/**This agent generates and updates the hashes file of all the required hosts
 * and returns to the home host to store them. Note that this agent needs to be
 * summoned by the Decrypter agent and needs to recieve from it the hashing algotithm,
 * the signer public key and the decrypter secret key encrypted with the home host public key.
@author Gatsu
@version 0.1
*/
public class Updater extends Agent {
 // Some widely used file names
 private String keysdir = "keys";
 private String targetsdir = "targetfiles";
 private String targethostsfilename = "updatehosts.list";
 private String zipfilename = "updatepackage.zip";

 private String pubkeyfilename = "pubkey.key";
 private String pubkeysfilename = "pubkey.sig";
 private String prikeyfilename = "prikey.key";
 private String prikeysfilename = "prikey.sig";
 //the algorithm used to compute the file hashes
 private String algorithm;
 //the agent name
 private String agentname;
 //the maximum waiting time for messages
 public long timeout = 10000;

 //a string that is equal to the not found hash
 public String notfoundhash;

 //general keys needed in both agents
 protected byte[] informdecrypterskeyb;
 protected PublicKey signerpubkey;
 protected transient SecretKey secretkey;
 //Global agent state
 protected String nexthostname;
 protected String homename;
 //the scheduler behaviour, that controls the agent execution flow
 private Behaviour scheduler;
 protected void setup(){
  Object[] args = getArguments();
  if (args.length == 3){
   //get the hashing algotithm
   algorithm = (String)args[0];
   //gets the signer public key
   signerpubkey=(PublicKey)args[1];
   //gets the decrypter secret key encrypted with the home host public key
   informdecrypterskeyb=(byte[])args[2];

   //agentname (not necesary but usefull if future modifications to allow several
   //coordinated updaters are desired)
   agentname = getLocalName();
   //save the home adress
   homename = here().getAddress();
   nexthostname = new String(homename);

   // Register languages and ontologies
   getContentManager().registerLanguage(new SLCodec());
   getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
   getContentManager().registerOntology(MobilityOntology.getInstance());
   //should register with the df, not necesary but could be used in the future to add new features
   try {
    //tryes to prepare a hash object to test if the algorithm parameter is correct and
    //prepares the file not found hash string (a string that thas the same length than
    //a hash but consists of a sequence of '-'
    Hash hash = new Hash(algorithm);
    int i = hash.length();
    char[] nullhashc = new char[i];
```

```
   for (--i; i >= 0; --i ){
    nullhashc[i] = '-';
   }
   notfoundhash = new String(nullhashc);

   System.out.println(getLocalName()+": Registering in the DF service");
   DFAgentDescription dfad = new DFAgentDescription();
   dfad.setName(getAID());
   ServiceDescription sd = new ServiceDescription();
   sd.setType("Updater");
   sd.setName(getName());
   sd.setOwnership("UDL");
   dfad.addServices(sd);
   DFService.register(this, dfad);
  } catch (Exception e) {
   e.printStackTrace();
   System.err.println(getLocalName()+": ERROR during the agent setup");
   doDelete();
  }
  //adds the main behaviour that schedules all the behaviours
  scheduler = new SchedulerBehaviour(this);
  addBehaviour( scheduler );
 }else{
  System.err.println(getLocalName()+": ERROR, wrong arguments recieved from the Decrypter");
  doDelete();
 }
}
/** Controls and schedules all behaviours in the main agent depending on the current state*/
class SchedulerBehaviour extends FSMBehaviour{
 public final int NOTAVAIABLE = -2;
 public final int FAILED = -1;
 public final int SUCCEDED = 0;
 public final int FINISHED = 2;
 private Location targethostl;
 private Location previoushostl;

 private transient PublicKey previoushostpublickey;
 private transient PrivateKey ownhostprivatekey;
 private transient PackagingTools pt;
 private byte[] previoushostpubkey;
 private byte[] previoushostpubkeysig;
 private boolean returnhome = true;
 public SchedulerBehaviour (Agent a){
  super(a);
  ChoseNextHostBehaviour cnhb = new ChoseNextHostBehaviour(a, this);
  registerFirstState(cnhb, "chosing next host");

  MigrationBehaviour mb = new MigrationBehaviour(a, this);
  registerState(mb, "migrating");

  SpawnHelperBehaviour shb = new SpawnHelperBehaviour(a, this);
  registerState( shb, "spawning helper at previous host");

  PrepareKeysBehaviour pkb = new PrepareKeysBehaviour(a, this);
  registerState( pkb, "preparing keys");

  ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
  request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
  request.addReceiver(new AID(getLocalName()+" helper", AID.ISLOCALNAME));
  request.setContent("Send me the package please");
  RecievePackageBehaviour rpb =new RecievePackageBehaviour(a, request, this);
  registerState( rpb, "recieving package from helper");

  PrepareReportBehaviour prb = new PrepareReportBehaviour(a, this);
  registerState(prb, "reporting");

  //Transitions
  registerDefaultTransition("chosing next host", "migrating");
```

```
  registerTransition("migrating", "chosing next host", NOTAVAIABLE);
  registerTransition("migrating", "spawning helper at previous host", SUCCEDED);

  registerDefaultTransition("spawning helper at previous host", "preparing keys");
  registerDefaultTransition("preparing keys", "recieving package from helper");
  registerDefaultTransition("recieving package from helper", "reporting");
  registerDefaultTransition("reporting", "chosing next host");
  scheduleFirst();
 }
}
/** Gets the next host in the list in order to migrate*/
class ChoseNextHostBehaviour extends OneShotBehaviour{
 int returnvalue;
 SchedulerBehaviour sb;
 public ChoseNextHostBehaviour(Agent a, SchedulerBehaviour sb){
  super(a);
  this.sb=sb;
 }
 public void action(){
  try{
   System.out.println(myAgent.getLocalName()+": chosing next host");
   returnvalue = sb.SUCCEDED;
   //if the target hosts file doesn't exist, initialize the packagingtools for this host
   //and unpack the targets file
   File targethostsfile = new File(targethostsfilename);
   if (!targethostsfile.canRead()){
    sb.pt = new PackagingTools(zipfilename, "unpack");
    sb.pt.unpackFileTo(".", targethostsfilename);
   }
   //prepares the hosts reader
   BufferedReader hosts_buffer = new BufferedReader(new FileReader(targethostsfilename));
   //at home with no host read
   if(nexthostname.equals(homename)){
    sb.previoushostl=myAgent.here();
    //read a host
    nexthostname = hosts_buffer.readLine();
    //the hosts file is empty!
    if (nexthostname == null){
     System.out.println(getLocalName()+": ERROR, The hosts file is empty");
     myAgent.doDelete();
    }
   }
   //if a hostname has already been read
   else{
    //Searching for the current read host in the hostsfile
    String readhostname;
    do{
     readhostname = hosts_buffer.readLine();
    }while(!nexthostname.equals(readhostname) && readhostname!=null);
    //read the next host
    nexthostname = hosts_buffer.readLine();
    if(nexthostname == null){
     nexthostname = new String(homename);
     sb.returnhome = true;
    }
   }
   hosts_buffer.close();
  }catch(Exception e){
   //some kind of rare error, return value should be changed
   returnvalue = sb.FAILED;
   e.printStackTrace();
  }
 }
}
/** Tryes to migrate to the next host*/
class MigrationBehaviour extends OneShotBehaviour{
 int returnvalue;
 SchedulerBehaviour sb;
```

```
public MigrationBehaviour(Agent a, SchedulerBehaviour sb){
 super(a);
 this.sb=sb;
}
public void action(){
 try{
  System.out.println(getLocalName()+": Migrating to "+nexthostname);
  returnvalue=sb.SUCCEDED;
  //Queryes the ams in order to get the possible destinations
  //Fills the message fields
  ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
  msg.addReceiver(getAMS());
  msg.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
  msg.setOntology(MobilityOntology.NAME);
  msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
  Action action = new Action();
  action.setActor(getAMS());
  action.setAction(new QueryPlatformLocationsAction());
  getContentManager().fillContent(msg, action);
  send(msg);

  //Receive response from AMS
  MessageTemplate mt = MessageTemplate.and(
    MessageTemplate.MatchSender(getAMS()),
    MessageTemplate.MatchPerformative(ACLMessage.INFORM));
  mt = MessageTemplate.and( mt, MessageTemplate.MatchOntology(MobilityOntology.NAME));
  mt = MessageTemplate.and( mt, MessageTemplate.MatchProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST));
  mt = MessageTemplate.and( mt, MessageTemplate.MatchLanguage(FIPANames.ContentLanguage.FIPA_SL0) );
  msg = blockingReceive(mt, timeout); //waiting at most timeout miliseconds
  if (msg != null){
   Result result = (Result)getContentManager().extractContent(msg);
   jade.util.leap.Iterator it = result.getItems().iterator();
   //tryes to match the location with the next one in the hosts file
   boolean found=false;
   while (it.hasNext() && !found) {
    sb.targethostl = (Location)it.next();
    found = sb.targethostl.getAddress().equals(nexthostname);
   }
   if (!found){
    //error, return value should be changed
    returnvalue=sb.NOTAVAIABLE;
   }
   else{
    //unpack the next host public key
    sb.pt.unpackFileTo(".", nexthostname+".sig");
    sb.pt.unpackFileTo(".", nexthostname+".key");
    sb.pt.finishUnpacking();
    //store the current host public key and its signature (will be checked later)
    FileInputStream pubkeyis = new FileInputStream(pubkeyfilename);
    FileInputStream pubkeysignis = new FileInputStream(pubkeysfilename);
    if (pubkeyis.available()>2048 || pubkeysignis.available()>2048){
     returnvalue=sb.FAILED;
    }
    else{
     sb.previoushostpubkey = new byte[pubkeyis.available()];
     sb.previoushostpubkeysig = new byte[pubkeysignis.available()];
     pubkeyis.read( sb.previoushostpubkey );
     pubkeysignis.read( sb.previoushostpubkeysig );
     pubkeyis.close();
     pubkeysignis.close();
     myAgent.doMove(sb.targethostl);
    }
   }
  }
  //no response from the ams
  else{
   returnvalue = sb.FAILED;
  }
```

```
  }catch(Exception e){
   //some kind of rare error, return value should be changed
   e.printStackTrace();
   returnvalue = sb.FAILED;
  }
 }
 public int onEnd(){
  return returnvalue;
 }
}
/** Spawn a helper agent that helps the main agent in the update process*/
class SpawnHelperBehaviour extends OneShotBehaviour{
 int returnvalue;
 SchedulerBehaviour sb;
 public SpawnHelperBehaviour(Agent a, SchedulerBehaviour sb){
  super(a);
  this.sb=sb;
 }
 public void action(){
  try{
   System.out.println(getLocalName()+": Spawning the Helper Agent");
   returnvalue=sb.SUCCEDED;
   myAgent.doClone(sb.previoushostl, getLocalName()+" helper");
   //updates the previous host
   sb.previoushostl=sb.targethostl;
  }catch (Exception e){
   returnvalue = sb.FAILED;
   e.printStackTrace();
  }
 }
 public int onEnd(){
  return returnvalue;
 }
}
/** Prepares the cryptographic keys needed to operate on this host*/
class PrepareKeysBehaviour extends OneShotBehaviour{
 int returnvalue;
 SchedulerBehaviour sb;
 public PrepareKeysBehaviour(Agent a, SchedulerBehaviour sb){
  super(a);
  this.sb=sb;
 }
 public void action(){
  try{
   System.out.println(myAgent.getLocalName()+": Preparing the required keys");
   returnvalue=sb.SUCCEDED;
   //read the private key
   FileInputStream prikeyis = new FileInputStream(prikeyfilename);
   FileInputStream prikeysignis = new FileInputStream(prikeysfilename);
   if (prikeyis.available()>2048 || prikeysignis.available()>2048){
    returnvalue = sb.FAILED;
   }
   else{
    byte[] sprikey = new byte[prikeyis.available()];
    byte[] prikeysignature = new byte[prikeysignis.available()];

    prikeyis.read( sprikey );
    prikeysignis.read( prikeysignature );
    //Warning! reads the entire file (it should be a small file)
    //checked before.

    //Checks if the private key is signed by the signer authority
    Signature rsasignature = Signature.getInstance("SHA1withRSA");
    rsasignature.initVerify(signerpubkey);
    rsasignature.update(sprikey);
    boolean privateverifyes = rsasignature.verify(prikeysignature);
    if (privateverifyes){
     //The key is correctly signed, parse it
```

```
     PKCS8EncodedKeySpec priKeySpec = new PKCS8EncodedKeySpec( sprikey );
     KeyFactory keyfactory = KeyFactory.getInstance("RSA");
     sb.ownhostprivatekey = keyfactory.generatePrivate(priKeySpec);
     prikeyis.close();
     prikeysignis.close();
    }
    //Checks if the public key is signed by the signer authority
    //Update and verify the data
    rsasignature.update(sb.previoushostpubkey);
    boolean publicverifyes = rsasignature.verify(sb.previoushostpubkeysig);
    if (publicverifyes){
     //The key is correctly signed, parse it
     X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec( sb.previoushostpubkey );
     KeyFactory keyfactory = KeyFactory.getInstance("RSA");
     sb.previoushostpublickey = keyfactory.generatePublic(pubKeySpec);
    }
    if(!privateverifyes || !publicverifyes){
     returnvalue = sb.FAILED;
    }
   }
  }catch (Exception e){
   returnvalue = sb.FAILED;
   e.printStackTrace();
  }
 }
 public int onEnd(){
  return returnvalue;
 }
}
/** Recieves the packaged files from the helper*/
class RecievePackageBehaviour extends AchieveREInitiator{
 int returnvalue;
 SchedulerBehaviour sb;
 transient Signature rsasignature;
 ACLMessage request;
 public RecievePackageBehaviour(Agent a, ACLMessage request, SchedulerBehaviour sb){
  super(a, request);
  this.sb=sb;
  this.request = request;
 }
 //When the helper agrees to transmit the file, the original should create the file
 //in the current host
 protected void handleAgree(ACLMessage agree){
  try{
   System.out.println(myAgent.getLocalName()+": Agreed to recieve the package");
   byte[] esecretkey = (byte[])agree.getContentObject();
   //now, we should decrypt the secret key, and use it to decrypt the package
   Cipher c1 = Cipher.getInstance("RSA");
   c1.init(Cipher.UNWRAP_MODE, sb.ownhostprivatekey);
   SecretKey secretkey = (SecretKey)c1.unwrap(esecretkey, "AES", Cipher.SECRET_KEY);

   c1 = Cipher.getInstance("AES");
   c1.init(Cipher.DECRYPT_MODE, secretkey);
   FileOutputStream zipfileos = new FileOutputStream(zipfilename);
   MessageTemplate filemessagetemplate = MessageTemplate.and(
     MessageTemplate.MatchSender(new AID(getLocalName()+" helper", AID.ISLOCALNAME)),
     MessageTemplate.MatchPerformative(ACLMessage.UNKNOWN));
   MessageTemplate informmessagetemplate = MessageTemplate.and(
     MessageTemplate.MatchSender(new AID(getLocalName()+" helper", AID.ISLOCALNAME)),
     MessageTemplate.MatchPerformative(ACLMessage.INFORM));
   ACLMessage recieve;
   byte[] cipheredchunk;
   byte[] filechunk;
   rsasignature = Signature.getInstance( "SHA1withRSA" );
   rsasignature.initVerify(sb.previoushostpublickey);
   boolean shouldinform = false;
   do{
    //blocks and awaits for incoming messages
```

```
    recieve = blockingReceive(timeout);
    //if it is a file chunk
    if(filemessagetemplate.match(recieve)){
     cipheredchunk = (byte[])recieve.getContentObject();
     rsasignature.update(cipheredchunk);
     zipfileos.write(c1.update(cipheredchunk));
    }
    else if(informmessagetemplate.match(recieve)){
     //finishes the decryption process
     zipfileos.write(c1.doFinal());
     zipfileos.close();
     //should put back the message to the recieve queue in order to allow the
     //conversation to continue
     myAgent.putBack(recieve);
     shouldinform = true;
    }
    else{
     //a strange message has been recieved, it has been read, so it is no longer
     //in the queue
     //IF THE AGENT IS EXTENDED WITH NEW FEATURES THAT REQUIRE MESSAGING
     //PUTTING BACK THE READED MESSAGE WOULD PROBABLY BE NEEDED
    }
   //If the transmission times out or a inform message is recieved, it stops waiting for
   //new messages
   }while(recieve != null && shouldinform ==false);
   if (recieve == null){
    System.err.println(getLocalName()+": Message waiting timed out");
    returnvalue = sb.FAILED;
   }
  }catch(Exception e){
   returnvalue = sb.FAILED;
   e.printStackTrace();
  }
 }
 protected void handleInform(ACLMessage inform) {
  try{
   System.out.println(myAgent.getLocalName()+": Recieved the inform message");
   if(!rsasignature.verify((byte[])inform.getContentObject())){
    System.out.println(myAgent.getLocalName()+": Signatures missmatch");
    returnvalue = sb.FAILED;
   }
  }catch(Exception e){
   returnvalue = sb.FAILED;
   e.printStackTrace();
  }
 }
 protected void handleRefuse(ACLMessage refuse){
  returnvalue = sb.FAILED;
 }
 public int onEnd(){
  //resets the behaviour in order to let it be reused
  this.reset(request);
  System.out.println(myAgent.getLocalName()+": Finished recieving the file");
  return returnvalue;
 }
}
/** Tells the Decrypter that the operation has finished when at home and the network
 * hashes update has been finished. This behaviour could be extended to perform other
 * tasks such as logging errors */
class PrepareReportBehaviour extends OneShotBehaviour{
 int returnvalue;
 SchedulerBehaviour sb;
 public PrepareReportBehaviour(Agent a, SchedulerBehaviour sb){
  super(a);
  this.sb=sb;
 }
 public void action(){
  returnvalue=sb.SUCCEDED;
```

```
  try{
   if(sb.returnhome == true && myAgent.here().getAddress().equals(homename)){
    System.out.println(getLocalName()+": Starting the report procedure");
    //decrypt the secret key (can be done only at the home platform)
    Cipher c1 = Cipher.getInstance("RSA");
    c1.init(Cipher.UNWRAP_MODE, sb.ownhostprivatekey);
    SecretKey idskey = (SecretKey)c1.unwrap(informdecrypterskeyb, "AES", Cipher.SECRET_KEY);
    //schedule the finishupdateBehaviour
    c1 = Cipher.getInstance("AES");
    c1.init(Cipher.ENCRYPT_MODE, idskey);

    ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
    request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
    request.addReceiver(new AID("Decrypter", AID.ISLOCALNAME));

    request.setContentObject(c1.doFinal("The update process has finished succesfully".getBytes()));
    //removes the schedulerBehaviour and queues the finishUpdaterBehaviour
    removeBehaviour(sb);
    addBehaviour(new FinishUpdaterBehaviour(myAgent, request, idskey));
   }
  }catch(Exception e){
   returnvalue = sb.FAILED;
   e.printStackTrace();
  }
 }
 public int onEnd(){
  return returnvalue;
 }
}
/** Sends an encrypted request to the Decrypter agent in order to finish the update*/
class FinishUpdaterBehaviour extends AchieveREInitiator{
 int returnvalue;
 transient Signature rsasignature;
 SecretKey idskey;
 transient boolean shouldinform = false;
 public FinishUpdaterBehaviour(Agent a, ACLMessage request, SecretKey idskey){
  super(a, request);
  this.idskey = idskey;
 }
 protected void handleRefuse(ACLMessage refuse){
  try{
   Cipher c1 = Cipher.getInstance("AES");
   c1.init(Cipher.DECRYPT_MODE, idskey);
   String cleartext = new String (c1.doFinal((byte[])refuse.getContentObject()));
   //if the refuse message is encrypted with a different secret key should suicide
   if(!cleartext.equals("REFUSE")){
    System.err.println(getLocalName()+": ERROR!, Somebody sent an incorrectly encrypted refuse message!");
    //delete the zip file and suicide
    File cleanfile = new File(zipfilename);
    cleanfile.delete();
    myAgent.doDelete();
   }
   //higly improbable but could happen due to man in the middle attacks
   else{
    System.err.println(getLocalName()+"WARNING, the Decrypter agent refused to process the data");
    //delete the zip file and suicide
    File cleanfile = new File(zipfilename);
    cleanfile.delete();
    myAgent.doDelete();
   }
  }catch(Exception e){
   e.printStackTrace();
   System.err.println(getLocalName()+": ERROR handling the refuse message!");
   //delete the zip file and suicide
   File cleanfile = new File(zipfilename);
   cleanfile.delete();
   myAgent.doDelete();
  }
```

```
  }
  protected void handleInform(ACLMessage inform){
   //should recieve a signed agree message
   try{
    System.out.println(myAgent.getLocalName()+": The Decrypter agent agreed to process the package");
   }catch(Exception e){
    //delete the zip file and suicide
    File cleanfile = new File(zipfilename);
    cleanfile.delete();
    myAgent.doDelete();
   }
  }
  public int onEnd(){
   myAgent.doDelete();
   return returnvalue;
  }
 }
//helper classes
 /** Controls and schedules all behaviours depending on the current state*/
 class HelperSchedulerBehaviour extends FSMBehaviour{
  public final int FAILED = -1;
  public final int SUCCEDED = 0;
  public Location targethostl;
  private transient PrivateKey ownhostprivatekey;
  public transient PublicKey nexthostpublickey;

  public HelperSchedulerBehaviour (Agent a){
   super(a);
   //Register state behaviours
   HashTargets htb =new HashTargets(a, this);
   registerFirstState(htb, "hashing");

   PrepareHelperKeysBehaviour phkb = new PrepareHelperKeysBehaviour(a, this);
   registerState(phkb, "preparing keys");

   PackageFilesBehaviour pfb = new PackageFilesBehaviour(a, this);
   registerState(pfb, "packaging");

   MessageTemplate mt = AchieveREResponder.createMessageTemplate(FIPANames.InteractionProtocol.FIPA_REQUEST);
   mt = MessageTemplate.and( mt, MessageTemplate.MatchContent("Send me the package please"));
   mt = MessageTemplate.and( mt, MessageTemplate.MatchSender(new AID(agentname, AID.ISLOCALNAME)));
   SendPackageBehaviour spb = new SendPackageBehaviour(a, mt, this);
   registerState(spb, "sending package");

   /* HACK */
   /* FIXME!: WARNING! DUE TO THE FACT THAT AN ACHIEVERERESPONSER BEHAVIOUR DOES NOT TERMINATE
    * (IT KEEPS WAITING FOR NEW MESSAGES) AND IS IMPOSSIBLE TO CHANGE THIS BEHAVIOUR WITHOUT
    * MODIFYING THE SOURCE CODE OF THE IMPLYED JADE CLASSES, THE FINISHHELPERBEHAVIOUR IS REALLY
    * ADDED WHEN THE SENDPACKAGEBEHAVIOUR SENDS THE INFORM MESSAGE, DOES NOT FOLLOW THE SCHEDULER
    * BEHAVIOUR.
    *
    * THE FINISHHELPERBEHAVIOUR SCHEDULING AND TRANSITIONS PERFORMED HERE SHOULD BE USED WHEN THE
    * BUG WILL BE FIXED AND THE BEHAVIOUR TERMINATES NORMALLY*/
   FinishHelperBehaviour fhb= new FinishHelperBehaviour(a, this);
   registerLastState(fhb, "finishing");
   //Transitions
   registerDefaultTransition("hashing", "preparing keys");
   registerDefaultTransition("preparing keys", "packaging");
   registerDefaultTransition("packaging", "sending package");

   registerDefaultTransition("sending package", "finishing");
   //start scheduling
   scheduleFirst();
  }
 }
 /** Hashes the files specifyed in the targets file and generates a hashes file. This behaviour
  * removes the targets file when it has finished */
 class HashTargets extends OneShotBehaviour{
```

```
int returnvalue;
HelperSchedulerBehaviour hsb;
public HashTargets(Agent a, HelperSchedulerBehaviour hsb){
 super(a);
 this.hsb=hsb;
}
public void action(){
 returnvalue=hsb.SUCCEDED;
 try{
  if(!(myAgent.here().getAddress().equals(homename)) ){
   System.out.println(getLocalName()+": Hashing targets");
   PackagingTools pt = new PackagingTools(zipfilename, "unpack");
   pt.unpackFileTo(".", myAgent.here().getAddress()+".targets" );
   pt.finishUnpacking();
   File targetsf = new File(myAgent.here().getAddress()+".targets");
   if ( targetsf.canRead() ){
    BufferedReader targetsbr = new BufferedReader(new FileReader(targetsf));
    FileWriter hashfilew = new FileWriter(myAgent.here().getAddress()+".hashes");
    String target = null;
    Hash hash = new Hash(algorithm);
    String hashstring = null;
    String hashfile = null;

    while( (target = targetsbr.readLine() ) != null) {
     File targetf = new File(target);
     if(targetf.canRead()){
      hashstring = hash.hashOfFile(target);
      hashfile = hash.hashedFilename();
     }
     else{
      hashstring = new String(notfoundhash);
      hashfile = target;
     }
     hashfilew.write( (hashstring + " " + hashfile + "\n"));
    }
    hashfilew.close();
    targetsbr.close();
    targetsf.delete();
    java.lang.System.gc(); //calls the garbage collector
   }
   else{
    returnvalue = hsb.FAILED;
   }
  }
 }catch(Exception e){
  returnvalue = hsb.FAILED;
  e.printStackTrace();
 }
}
public int onEnd(){
 return returnvalue;
}
}
/** Prepare the keys needed to encrypt the package and sign it*/
class PrepareHelperKeysBehaviour extends OneShotBehaviour{
 int returnvalue;
 HelperSchedulerBehaviour hsb;
 public PrepareHelperKeysBehaviour(Agent a, HelperSchedulerBehaviour hsb){
  super(a);
  this.hsb=hsb;
 }
 public void action(){
  try{
   System.out.println(myAgent.getLocalName()+": Preparing keys");
   returnvalue= hsb.SUCCEDED;
   FileInputStream prikeyis = new FileInputStream(prikeyfilename);
   FileInputStream prikeysignis = new FileInputStream(prikeysfilename);
```

```
  FileInputStream pubkeyis = new FileInputStream(nexthostname+".key");
  FileInputStream pubkeysignis = new FileInputStream(nexthostname+".sig");

  //they should be small files
  if (pubkeyis.available()>2048 || pubkeysignis.available()>2048 ||
      prikeyis.available()>2048 || prikeysignis.available()>2048 ){
   returnvalue = hsb.FAILED;
  }
  else{
   byte[] sprikey = new byte[prikeyis.available()];
   byte[] prikeysignature = new byte[prikeysignis.available()];

   byte[] spubkey = new byte[pubkeyis.available()];
   byte[] pubkeysignature = new byte[pubkeysignis.available()];
   prikeyis.read( sprikey );
   prikeysignis.read( prikeysignature );

   pubkeyis.read( spubkey ); //Warning! reads the entire file (it should be a small file)
   pubkeysignis.read( pubkeysignature ); //checked before.

   //Checks if the private key is signed by the signer authority
   Signature rsasignature = Signature.getInstance("SHA1withRSA");
   rsasignature.initVerify(signerpubkey);
   rsasignature.update(sprikey);
   boolean privateverifyes = rsasignature.verify(prikeysignature);
  //ownhostprivatekey
   if (privateverifyes){
    //The key is correctly signed, parse it
    PKCS8EncodedKeySpec priKeySpec = new PKCS8EncodedKeySpec( sprikey );
    KeyFactory keyfactory = KeyFactory.getInstance("RSA");
    hsb.ownhostprivatekey = keyfactory.generatePrivate(priKeySpec);
    prikeyis.close();
    prikeysignis.close();
   }

   //Checks if the public key is signed by the signer authority
   //Update and verify the data
   rsasignature.update(spubkey);
   boolean publicverifyes = rsasignature.verify(pubkeysignature);
  //nexthostpublickey
   if (publicverifyes){
    //The key is correctly signed, parse it
    X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec( spubkey );
    KeyFactory keyfactory = KeyFactory.getInstance("RSA");
    hsb.nexthostpublickey = keyfactory.generatePublic(pubKeySpec);
    pubkeyis.close();
    pubkeysignis.close();
   }
  //secretkey
   if (privateverifyes && publicverifyes){
    //now, we should create a secret key
    KeyGenerator keygen = KeyGenerator.getInstance("AES");
    secretkey = keygen.generateKey();
   }
   else{
    returnvalue = hsb.FAILED;
   }
  }
  }catch (Exception e){
   returnvalue = hsb.FAILED;
   e.printStackTrace();
  }
 }
 public int onEnd(){
  return returnvalue;
 }
}
/** Packages the newly created hashes file. If the agent is in the home platform this
```

```
 * behaviour does nothing*/
class PackageFilesBehaviour extends OneShotBehaviour{
 int returnvalue;
 HelperSchedulerBehaviour hsb;
 public PackageFilesBehaviour(Agent a, HelperSchedulerBehaviour hsb){
  super(a);
  this.hsb=hsb;
 }
 public void action(){
  try{
   System.out.println(getLocalName()+": Packaging the updated files");
   returnvalue= hsb.SUCCEDED;
   //Updates the package if not at home
   if(!homename.equals(myAgent.here().getAddress())){
    //prepares the replacements vector (only one replacement);
    Vector<String[]> replacevector = new Vector<String[]>();
    String[] replacement = {
     myAgent.here().getAddress()+".targets",
     myAgent.here().getAddress()+".hashes",
     myAgent.here().getAddress()+".hashes"
    };
    replacevector.add(replacement);
    PackagingTools pt = new PackagingTools(zipfilename, "replace files");
    pt.replaceFiles(replacevector);
    pt.finishPackaging();
   }
   //Otherwise the agent leaves the package untouched
  }catch(Exception e){
   returnvalue = hsb.FAILED;
   e.printStackTrace();
  }
 }
 public int onEnd(){
  return returnvalue;
 }
}
/** Encrypts signs and sends the package*/
class SendPackageBehaviour extends AchieveREResponder{
 int returnvalue;
 HelperSchedulerBehaviour hsb;
 MessageTemplate mt;
 public SendPackageBehaviour(Agent a, MessageTemplate mt, HelperSchedulerBehaviour hsb){
  super(a, mt);
  this.hsb=hsb;
  this.mt = mt;
 }
 protected ACLMessage prepareResponse(ACLMessage request){
  System.out.println(myAgent.getLocalName()+": Agreing to the send the file");
  ACLMessage response = request.createReply();
  response.setPerformative(ACLMessage.AGREE);
  //sends the secret key encrypted using the next host public key
  try{
   Cipher c1 = Cipher.getInstance("RSA");
   c1.init(Cipher.WRAP_MODE, hsb.nexthostpublickey);
   response.setContentObject(c1.wrap(secretkey));
  }catch(Exception e){
   returnvalue = hsb.FAILED;
   e.printStackTrace();
  }
  return response;
 }
 protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage response) {
  ACLMessage informdone;
  informdone = request.createReply();
  informdone.setPerformative(ACLMessage.INFORM);
  try{
   //starts the cipher with the secret key
   returnvalue= hsb.SUCCEDED;
```

```
   Cipher c1 = Cipher.getInstance("AES");
   c1.init(Cipher.ENCRYPT_MODE, secretkey);

   //starts reading, encrypting and signing the zipfile and sends the encrypted chunks
   //using aclmessages
   FileInputStream fis = new FileInputStream(zipfilename);
   ACLMessage filemessage= new ACLMessage(ACLMessage.UNKNOWN);
   filemessage.addReceiver(request.getSender());
   byte[] filechunk = new byte[8192];//8kb
   byte[] cipheredchunk;
   int readbytes;
   Signature rsasignature = Signature.getInstance( "SHA1withRSA" );
   rsasignature.initSign( hsb.ownhostprivatekey );
   while ( ( readbytes = fis.read( filechunk )) != -1){
    cipheredchunk = c1.update(filechunk, 0, readbytes);
    rsasignature.update( cipheredchunk );
    filemessage.setContentObject(cipheredchunk);
    send(filemessage);
   }
   //the last pass (padding and several operations needed to finish the encryption)
   cipheredchunk = c1.doFinal();
   rsasignature.update( cipheredchunk );
   filemessage.setContentObject(cipheredchunk);
   send(filemessage);

   System.out.println(myAgent.getLocalName()+": Finished sending the file, notificating");

   //inform and finish the protocol
   //sends the encrypted file signature in order to check it
   informdone.setContentObject(rsasignature.sign());
  }catch(Exception e){
   returnvalue = hsb.FAILED;
   e.printStackTrace();
  }
  //adding the hack
  addBehaviour( new HackAchieveREResponderBehaviour(myAgent, hsb));
  return informdone;
 }
 public int onEnd(){
  System.out.println(getLocalName()+": Finally finished the request protocol");
  return returnvalue;
 }
}
/** A nasty hack to Schedule the FinishHelperBehaviour. It seems to be impossible to end it normally,
 * it just keeps restarting the achieve rational effect protocol instead of finishing it*/
class HackAchieveREResponderBehaviour extends OneShotBehaviour{
 HelperSchedulerBehaviour hsb;
 public HackAchieveREResponderBehaviour(Agent a, HelperSchedulerBehaviour hsb){
  super (a);
  this.hsb=hsb;
 }
 public void action(){
  addBehaviour (new FinishHelperBehaviour(myAgent, hsb));
 }
}
/** Finish the helper lyfecicle. This behaviour deletes all the generated files that contain
 *  sensitive information*/
class FinishHelperBehaviour extends OneShotBehaviour{
 HelperSchedulerBehaviour hsb;
 public FinishHelperBehaviour(Agent a, HelperSchedulerBehaviour hsb){
  super(a);
  this.hsb=hsb;
 }
 public void action(){
  try{
   //Delete the unpacked files
   File cleanfile = new File(zipfilename);
   cleanfile.delete();
```

```
    cleanfile = new File(nexthostname+".sig");
    cleanfile.delete();
    cleanfile = new File(nexthostname+".key");
    cleanfile.delete();
    cleanfile = new File(myAgent.here().getAddress()+".targets");
    cleanfile.delete();
    cleanfile = new File(myAgent.here().getAddress()+".hashes");
    cleanfile.delete();
    //if not at home, should also delete the targets file
    if(!(myAgent.here().getAddress().equals(homename))){
     cleanfile = new File(targethostsfilename);
     cleanfile.delete();
    }
   }catch( Exception e){
    System.err.println(getLocalName()+": WARNING! Can't delete some of the update files");
    e.printStackTrace();
   }finally{
    myAgent.doDelete();
   }
  }
 }
 protected void beforeMove() {
 }
 /** Reregister the required languages and ontologies*/
 protected void afterMove() {
  try{
   System.out.println(getLocalName()+": Reregistering the required languages and ontologies");
   // Register languages and ontologies
   getContentManager().registerLanguage(new SLCodec());
   getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
   getContentManager().registerOntology(MobilityOntology.getInstance());
  }catch(Exception e){
   e.printStackTrace();
  }
 }
 protected void beforeClone() {
 }
 /** Changes the scheduler. Triggered in the clone when finished cloning*/
 protected void afterClone() {
  removeBehaviour(scheduler);
  scheduler = new HelperSchedulerBehaviour(this);
  addBehaviour(scheduler);
 }
 /** Deregisters the agent entry in the df (if the agent is not a helper).*/
 protected void takeDown(){
  try{
   if(getLocalName().equals(agentname)){
    DFService.deregister(this);
   }
   System.out.println(getLocalName()+": Terminating");
  }catch (Exception e){
   System.err.println("Error destroying the agent");
  }
 }
}
```

## A.4 PackagingTools.java

```
package ammaf.util;
import java.io.*;
import java.util.*;
import java.util.zip.*;
import java.text.Collator;
/** A simple class that performs the packaging operations part of the AMMAF project */
public class PackagingTools {
```

```
private boolean replacingfinished=false;
private boolean finished=false;
private String mode;
private String zippackagefilename;
private ZipOutputStream zipouts;
private ZipFile zipfile;
/** Constructor method. It starts the packaging.
 * @param zippackagefilename The zip file that should be created or unpacked
 * @param mode The execution mode, should be one of the following:
 * "add files", "new package", "replace files" or "unpack".
 * @throws Exception if there was any error*/
public PackagingTools(String zippackagefilename, String mode) throws Exception{
 this.mode=mode;
 this.zippackagefilename=zippackagefilename;
 try{
  this.init();
 }catch(Exception e){
  throw e;
 }
}
/** Called by the constructor method. Decides wich operations should be performed
 * and calls the necessary methods.
 * @throws Exception if there was any error*/
private void init() throws Exception{
 File zipf = new File(zippackagefilename);
 try{
  if ( mode.equals("add files") || mode.equals("new package") || mode.equals("replace files") ){
   zipouts = new ZipOutputStream( new FileOutputStream(zippackagefilename + ".tmp") );
   zipouts.setLevel(9); //maximum compression level
   if (mode.equals("add files")){
    if (zipf.canRead()){
     this.copyZipFile();
    }
    else{
     File tzipf = new File(zippackagefilename + ".tmp");
     tzipf.delete();
     finished = true;
     Exception e = new Exception();
     throw e;
    }
   }
  }
  if ( mode.equals("unpack") ){
   zipfile = new ZipFile(zippackagefilename);
  }
 }catch (Exception e){
  if ( mode.equals("add files") || mode.equals("new package") || mode.equals("replace files") ){
   File tzipf = new File(zippackagefilename + ".tmp");
   tzipf.delete();
  }
  finished=true;
  throw e;
 }
}
/** An operation that copyes the contents of an existing zip package to a new one.
 * @throws Exception if there was any error */

private void copyZipFile() throws Exception{
 try{
  ZipInputStream zis = new ZipInputStream(new BufferedInputStream(new FileInputStream(zippackagefilename)));
  ZipEntry ze = null;
  while((ze = zis.getNextEntry()) != null) {
   this.addISToPackage(zis, ze.getName());
  }
  zis.close();
 }catch (Exception e){
  throw e;
 }
```

```
}
/** An operation that copyes the contents of an existing zip package to a new one and replaces the
 * desired files. This method should be called only once with all the desired replacements and
 * must be called before adding new files to the package.
 * @param desiredreplacements the ordered by oldpackagefile name list of string arrays, uses a
 * collator to compare the strings locale wisely.
 * oldpackagefile desiredfile desiredfilepackagename
 * if the parameters desiredfile and desiredfilepackagename are null, the oldpackagedfile is removed
 * from the new package.
 * @throws Exception if ther was any error*/
public void replaceFiles(Vector<String[]> desiredreplacements) throws Exception{
 if (mode.equals("replace files") && !replacingfinished){
  int comparison;
  int matchcounter;
  boolean found=false;
  String[] replacement=null;
  String packagedname;
  // Prepares a collator in order to compare two strings using the default locale
  Collator mycollator = Collator.getInstance();

  ZipInputStream zis = new ZipInputStream(new BufferedInputStream(new FileInputStream(zippackagefilename)));
  ZipEntry ze = null;
  while((ze = zis.getNextEntry()) != null) {
   //try to match the current entry with a desired replacement
   matchcounter = 0;
   comparison = -1;
   packagedname = ze.getName();
   while( comparison < 0 && matchcounter < desiredreplacements.size()){
    replacement = desiredreplacements.get(matchcounter);
    if(replacement.length == 3){ //should contain 3 strings
     comparison = mycollator.compare( replacement[0], packagedname );
    }
    else{
     File tzipf = new File(zippackagefilename + ".tmp");
     tzipf.delete();
     finished=true;
     Exception e= new Exception();
     throw e;
    }
    matchcounter++;
   }
   //matchcounter was really one unit less
   matchcounter--;
   //matched
   if(comparison == 0 && desiredreplacements.size() > 0){
    //if both desired file and desired package names are not null, should perform
    //the replacement
    if( replacement[1]!=null && replacement[2]!=null){
     //prepare the desired file
     FileInputStream fis = new FileInputStream(replacement[1]);
     this.addISToPackage(fis, replacement[2]);
     fis.close();
     //remove the replacement from the list and updates the matchcounter
     desiredreplacements.remove(matchcounter--);
    }
    //if both names are null it skips the file (does not add it to the new package)
    else if ( replacement[1]==null && replacement[2]==null){
     //remove the replacement from the list and updates the matchcounter
     desiredreplacements.remove(matchcounter--);
    }
    //one name is null, throwing an exception
    else{
     File tzipf = new File(zippackagefilename + ".tmp");
     tzipf.delete();
     finished=true;
     Exception e= new Exception();
     throw e;
    }
```

```
   }
   else{
    //adds the file to the package normally
    addISToPackage(zis, ze.getName());
   }
  }
  zis.close();
  replacingfinished=true;
  if(desiredreplacements.size()>0){
   //some replacements could not be done because the desired files were not present in
   //the package
   File tzipf = new File(zippackagefilename + ".tmp");
   tzipf.delete();
   finished=true;
   Exception e= new Exception();
   throw e;
  }
 }
}
/** A private operation that adds the contents of the given input stream to the package
 * @param is the input stream that is going to be packaged
 * @param packagedfilename the name given to the packaged input stream
 * @throws Exception if there was any error */
private void addISToPackage(InputStream is, String packagedfilename) throws Exception{
 try{
  byte[] buffer = new byte[8192]; //8kB
  ZipEntry ze = new ZipEntry (packagedfilename);
  zipouts.putNextEntry(ze);
  int readbytes=0;
  while ( (readbytes = is.read( buffer )) != -1){
   zipouts.write( buffer, 0, readbytes );
  }
  zipouts.closeEntry();
 }catch (Exception e){
  throw e;
 }
}
/** If this instance is in packaging mode, this method adds a new file to the package
 * @param tobepackagedfilename The name of the file that is going to be packaged
 * @param packagedfilename The name that the file will have in the package
 * @throws Exception if there was any error
 * @throws ZipException if the mode is incorrect or the operation has already finished */
public void addToPackage(String tobepackagedfilename, String packagedfilename) throws Exception, ZipException{
 FileInputStream fis = null;
 try{
  if( (mode.equals("add files") || mode.equals("new package") || (mode.equals("replace files") &&
      replacingfinished)) && !finished){
   fis = new FileInputStream (tobepackagedfilename);
   this.addISToPackage(fis, packagedfilename);
   fis.close();
  }
  else{
   ZipException ze = new ZipException();
   throw ze;
  }
 }catch (Exception e){
  fis.close();
  File tzipf = new File(zippackagefilename + ".tmp");
  tzipf.delete();
  throw e;
 }
}
/** Finishes the packaging process. Closes the zipoutputstream and renames the temporaly file to
 * its original name. This method also sets the finished flag to true in order to ban incorrect
 * reutilization of the class.
 * @throws Exception if there was any error
 * @throws ZipException if the mode is incorrect or the operation has already finished */
public void finishPackaging() throws Exception{
```

```
 try{
  if( (mode.equals("add files") || mode.equals("new package") || mode.equals("replace files")) && !finished){
   zipouts.close();
   File zpfnt = new File(zippackagefilename + ".tmp");
   File zpfn = new File(zippackagefilename);
   zpfn.delete();
   zpfnt.renameTo(zpfn);
   finished=true;
  }
  else{
   finished = true;
   File tzipf = new File(zippackagefilename + ".tmp");
   tzipf.delete();
   ZipException ze = new ZipException();
   throw ze;
  }
 }catch (Exception e){
  File tzipf = new File(zippackagefilename + ".tmp");
  tzipf.delete();
  finished = true;
  throw e;
 }
}
/** Unpacks the contents of the zip package to the given directory
 * @param unpackdir the target directory where all the files will be unpacked
 * @throws Exception if there was any error
 * @throws IOException if unpackdir does not point to a valid directory
 * @throws ZipException if the mode is incorrect or the operation has already finished*/
public void unpackTo(String unpackdir) throws Exception, IOException, ZipException{
 try{
  if(mode.equals("unpack") && !finished){
   File unpackdirf = new File (unpackdir);
   if(unpackdirf.isDirectory()){
    InputStream is;
    //ZipFile zipfile = new ZipFile(zippackagefilename);
    ZipEntry ze;
           Enumeration enu = zipfile.entries();
    while( enu.hasMoreElements() ){
     ze = (ZipEntry) enu.nextElement();
     is = zipfile.getInputStream(ze);
     this.writeISToFile(is, unpackdir + File.separator + ze.getName() );
    }
   }
   else{
    IOException ioe = new IOException();
    throw ioe;
   }
  }
  else{
   finished = true;
   ZipException ze = new ZipException();
   throw ze;
  }
 }catch(Exception e){
  throw e;
 }
}
/** Private operation that writes the contents of the given input stream to the target file
 * @param is The input stream that is going to be written
 * @param filename The name of the file that is going to be created */
private void writeISToFile(InputStream is, String filename) throws Exception{
 try{
  FileOutputStream fos = new FileOutputStream(filename);
  byte[] buffer = new byte[8192];
  int readbytes=0;
  while ( (readbytes = is.read( buffer )) != -1){
   fos.write( buffer, 0, readbytes );
  }
```

```
  fos.close();
 }catch (Exception e){
  throw e;
 }
}
/** Searches for the desired file in the package and unpacks to the target directory
 * @param wantedfile The file held in the package that must be unpacked
 * @param unpackdir The target directory where the wanted file must be unpacked
 * @throws Exception if there was any error
 * @throws FileNotFoundException if the wanted file was not found
 * @throws IOException if the unpackdir is not a valid directory
 * @throws ZipException if the mode is incorrect or the operation has already finished */
public void unpackFileTo(String unpackdir, String wantedfile) throws Exception,
                        FileNotFoundException, IOException, ZipException{
 try{
  if(mode.equals("unpack") && !finished){
   File unpackdirf = new File (unpackdir);
   if(unpackdirf.isDirectory()){
    InputStream is;
    //ZipFile zipfile = new ZipFile(zippackagefilename);
    ZipEntry ze;
    ze=zipfile.getEntry(wantedfile);
    if (ze != null){
     is = zipfile.getInputStream(ze);
     this.writeISToFile(is, unpackdir + File.separator +ze.getName() );
    }
    else{
     FileNotFoundException fnfe = new FileNotFoundException();
     throw fnfe;
    }
   }
   else{
    IOException ioe = new IOException();
    throw ioe;
   }
  }
  else{
   finished = true;
   ZipException ze = new ZipException();
   throw ze;
  }
 }catch(Exception e){
  finished = true;
  throw e;
 }
}
/** Searches for the desired file in the package.
 * @param wantedfile The file whose presence in the package is checked
 * @throws ZipException if the mode is incorrect or the operation has already finished or
 * if there was a problem reading the zipfile
 * @return a bolean whose value is true if the wantedfile has been found, false otherwise*/
public boolean hasFile(String wantedfile) throws ZipException{
 boolean present = false;
 try{
  if(mode.equals("unpack") && !finished){
   ZipEntry ze;
   ze=zipfile.getEntry(wantedfile);
   present = (ze!=null);
  }
  else{
   ZipException ze = new ZipException();
   throw ze;
  }
 }catch(Exception e){
  ZipException ze = new ZipException();
  throw ze;
 }
 return present;
```

```
 }
 /** Finishes the unpacking process. Closes the zipfile and sets the finished flag to true in
  * order to ban incorrect reutilization of the class.
  * @throws Exception if there was any error
  * @throws ZipException if the mode is incorrect or the operation has already finished */
 public void finishUnpacking() throws Exception{
  try{
   if(mode.equals("unpack") && !finished){
    zipfile.close();
    finished = true;
   }
   else{
    finished = true;
    ZipException ze = new ZipException();
    throw ze;
   }
  }catch (Exception e){
   finished = true;
   throw e;
  }
 }
}
```

## A.5 Hash.java

```
/**file digestion
* @author Gatsu*/
package ammaf.util;
import java.security.*;
import java.io.*;
/** Digests files using the desired algorithm (default is SHA-256)*/
public class Hash {
 private String algorithm = "SHA-256";
 private MessageDigest md;
 private File hashedfile;
 private int length;
 /** Initializes the Hash digestion, initializing the underlying MessageDigest object
  * @throws NoSuchAlgorithmException if the requested algorithm doesn't exist*/
 public Hash () throws NoSuchAlgorithmException {
  try{
   setAlgorithm(this.algorithm);
  }catch (NoSuchAlgorithmException nsae) {
   throw nsae;
  }
 }
 public Hash (String algorithm) throws NoSuchAlgorithmException {
  try{
   setAlgorithm(algorithm);
  }catch (NoSuchAlgorithmException nsae) {
   throw nsae;
  }
 }
 /** Performs the digestion and computes it's hex string. Note that one Hash object can
  * perform several digestions calling this method
  * @param filename A string that defines the file that is going to be digested
  * @return A string that contains the hash of the filename file in hex format
  * @throws IOException if there was an error reading the target file*/
 public String hashOfFile (String filename) throws IOException {
  DigestInputStream dis;
  hashedfile = new File (filename);
  if ( hashedfile.canRead() ){
   try{
    FileInputStream fis = new FileInputStream (hashedfile);
    dis = new DigestInputStream (fis, md); //Associates the message digest class with
                                           //the read operation of the file
    byte [] read_buffer = new byte[8192]; //8kB (aproximated maximum block size)
```

```
    while ( dis.read(read_buffer, 0, 8192) != -1 );
    byte[] raw = md.digest();
    String output_hash = UtilUnsigned.byteToHex(raw);
    fis.close();
    return (output_hash);
   }catch (IOException ioe){
    ioe= new IOException();
    throw ioe;
   }
  }
  else {
   IOException ioe = new IOException();
   throw ioe;
  }
 }
 /** Returns the complete file name of the last file digested in a Canonical way
  * (i.e standart, with the full path)
  * @return A string that contains the filename of the last digested file in a canonical way
  * @throws IOException if no file has been digested or there was an error getting the path*/
 public String hashedFilename () throws IOException {
  try{
   return (hashedfile.getCanonicalPath());
  }catch (IOException ioe){
   throw ioe;
  }
 }
 /** Returns the hash length.
  * @return An int that means the hash length in bytes*/
 public int length (){
  return length;
 }
 /** Changes the hashing algorithm to the desired one. The desired algorithm  should exist
  * and should be correct in order to avoid exceptions when computing the hash
  * @param algorithm A string that defines the file that is going to be digested*/
 public void setAlgorithm (String algorithm) throws NoSuchAlgorithmException{
  try{
   this.algorithm=new String(algorithm);
   md = MessageDigest.getInstance (algorithm);
   length = md.getDigestLength()*2; //expressed in hexadecimal the lenght doubles the amount bytes
  }catch (NoSuchAlgorithmException nsae) {
   throw nsae;
  }
 }
}
```

# B Documentació Javadoc del codi font

En aquest apèndix s'inclou la documentació javadoc de cada una de les classes. En el CD-ROM adjunt es pot trobar l'arbre complet de documentació Javadoc en html. La classe UtilUnsigned no disposa de documentació ja que es tracta d'una classe externa.

## B.1 Signer.java

```
================================================================================
***** ammaf
Class Signer *****
java.lang.Object
  [extended by ]jade.core.Agent
      [extended by ]ammaf.Signer
  All Implemented Interfaces:
      jade.core.TimerListener, java.io.Serializable, java.lang.Runnable
================================================================================
  public class Signer
  extends jade.core.Agent
This agent generates and signs the keypairs of all the to be checked hosts and
returns the home host to store its public key and signature. Note that this
agent needs a target hosts file named "hosts.list" to know where should it move
to. The "hosts.list" file should consist in a simple text file that contains
host names separated by carry returns.
  See Also:
      Serialized Form
================================================================================

|Nested Class Summary                                                          |
|(package private)  class|Signer.AskForMigration                               |
|                        |          This behaviour querries the ams for       |
|                        |possible destinations and searches the destination  |
|                        |found in                                            |
|(package private)  class|Signer.GenerateHostKeypair                          |
|                        |          This behaviour generates the host keypair.|
|(package private)  class|Signer.Migration                                    |
|                        |          A simple behaviour that allows the agent  |
|                        |to move to next host when cloned                    |
|(package private)  class|Signer.PackageFiles                                 |
|                        |          This behaviour creates the to be          |
|                        |transfered package.                                 |
|(package private)  class|Signer.PrepareSigningKeypair                        |
|                        |          A behaviour that prepares (loads or       |
|                        |generates) the master keypair if the hosts file     |
|                        |contains at least one host.                         |
|(package private)  class|Signer.SignHostKeypair                              |
|                        |          This behaviour generates the host keypair |
```

| | |signature. |
|(package private) class|Signer.StoreIncomingFile |
| | | Recieves and stores the remote file. |
|(package private) class|Signer.StoreKeysInHost |
| | | This behaviour stores the keypair and its |
| | |signature at the host. |
|(package private) class|Signer.TransferReply |
| | | An AchieveREResponder behaviour that |
| | |handles the file transfer at the helper agent. |
|(package private) class|Signer.TransferRequest |
| | | An AchieveREInitiator behaviour that |
| | |handles the file transfer at the original |

|Nested classes/interfaces inherited from class jade.core.Agent|
|jade.core.Agent.Interrupted |

|Field Summary |
| protected boolean|checkcurrent |
| | |
| protected java.lang.String|checkhostname |
| | |
| protected boolean|error |
| | |
| protected java.lang.String|homename |
| | |
| protected byte[]|hprivate |
| | |
| protected byte[]|hpublic |
| | |
| protected byte[]|priksignature |
| | |
| protected byte[]|pubksignature |
| | |
|protected java.security.KeyPair|signingkeypair |
| | Key variables|

|Fields inherited from class jade.core.Agent |
|AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED,|
|AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN |

|Constructor Summary|
|Signer() |
| |

|Method Summary |

|protected  void|afterClone()                                                    |
|               |          Deletes the hosts file and prepares for the          |
|               |incoming transfer adding a TransferReplyBehaviour.             |
|protected  void|afterMove()                                                     |
|               |                                                               |
|protected  void|beforeClone()                                                   |
|               |          Sets the cloned variable to true.                    |
|protected  void|beforeMove()                                                    |
|               |                                                               |
|protected  void|setup()                                                         |
|               |          Initialization method                                |
|protected  void|takeDown()                                                      |
|               |          Deregisters the agent entry in the df (if the agent|
|               |is not a helper).                                              |

|Methods inherited from class jade.core.Agent                                  |
|addBehaviour, blockingReceive, blockingReceive, blockingReceive,              |
|blockingReceive, changeStateTo, clean, doActivate, doClone, doDelete, doMove,|
|doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, |
|getArguments, getContainerController, getContentManager, getCurQueueSize,    |
|getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject,         |
|getProperty, getQueueSize, getState, here, notifyChangeBehaviourState,        |
|notifyRestarted, postMessage, putBack, putO2AObject, receive, receive,        |
|removeBehaviour, restartLater, restore, restoreBufferedState, run, send,      |
|setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents,         |
|setQueueSize, waitUntilStarted, write                                         |

|Methods inherited from class java.lang.Object                              |
|clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,|
|wait, wait, wait                                                           |

|Field Detail|
**** signingkeypair ****
protected java.security.KeyPair signingkeypair
      Key variables
================================================================================
**** hpublic ****
protected transient byte[] hpublic
================================================================================
**** hprivate ****
protected transient byte[] hprivate
================================================================================
**** pubksignature ****
protected transient byte[] pubksignature
================================================================================
**** priksignature ****
protected transient byte[] priksignature

```
================================================================================
**** error ****
protected boolean error
================================================================================
**** checkcurrent ****
protected boolean checkcurrent
================================================================================
**** checkhostname ****
protected java.lang.String checkhostname
================================================================================
**** homename ****
protected java.lang.String homename

|Constructor Detail|
**** Signer ****
public Signer()

|Method Detail|
**** setup ****
protected void setup()
      Initialization method
        Overrides:
            setup in class jade.core.Agent
================================================================================
**** beforeMove ****
protected void beforeMove()
        Overrides:
            beforeMove in class jade.core.Agent
================================================================================
**** afterMove ****
protected void afterMove()
        Overrides:
            afterMove in class jade.core.Agent
================================================================================
**** beforeClone ****
protected void beforeClone()
      Sets the cloned variable to true. Triggered in both the agent and the
      clone before cloning
        Overrides:
            beforeClone in class jade.core.Agent
================================================================================
**** afterClone ****
protected void afterClone()
      Deletes the hosts file and prepares for the incoming transfer adding a
      TransferReplyBehaviour. Triggered in the clone when finished cloning
        Overrides:
            afterClone in class jade.core.Agent
================================================================================
**** takeDown ****
protected void takeDown()
```

```
      Deregisters the agent entry in the df (if the agent is not a helper).
        Overrides:
            takeDown in class jade.core.Agent
================================================================================
```

## B.2 Decrypter.java

```
================================================================================
***** ammaf
Class Decrypter *****
java.lang.Object
  [extended by ]jade.core.Agent
      [extended by ]ammaf.Decrypter
  All Implemented Interfaces:
      jade.core.TimerListener, java.io.Serializable, java.lang.Runnable
================================================================================
  public class Decrypter
  extends jade.core.Agent
This agent starts the system and when the whole process has been completed, it
decrypts and reports the results. Needs the following things: The parameter
update to start the update process (Done to structure and ease future
modifications to the system). A signed keypair in every host, a master keypair
in the current directory and the public keys and its signatures in the
subdirectory keys (so first the agent system should run the Signer agent). A
updatehosts.list file that contains the hostnames of the to be checked host
separated by carry returns. A subdirectory called targetsdir that contains a
targets file for each host separated by carry returns that needs to be named
[hostname].targets
  See Also:
      Serialized Form
================================================================================

|Nested Class Summary                                                          |
|(package private)  class|Decrypter.PackageUpdaterFiles                        |
|                        |          Gathers all the needed files and creates a|
|                        |package file.                                        |
|(package private)  class|Decrypter.ProcessUpdate                              |
|                        |          Awaits for the completion of the update    |
|                        |process and attends the request to store the results|
|(package private)  class|Decrypter.ReportUpdateResults                        |
|                        |          Unpacks the generated hashes files         |
|(package private)  class|Decrypter.Scheduler                                  |
|                        |          Controls and schedules all behaviours      |
|(package private)  class|Decrypter.StartUpdater                               |
|                        |          Prepares the Decryption keypair and passes|
|                        |its public part to the updater.                      |
```

|Nested classes/interfaces inherited from class jade.core.Agent|
|jade.core.Agent.Interrupted |

|Field Summary |
|protected boolean|error |
| | |
|protected java.lang.String|mode |
| | |
|protected java.security.Key|privatekey |
| | |
|protected javax.crypto.SecretKey|secretkey |
| | |

|Fields inherited from class jade.core.Agent |
|AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED,|
|AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN |

|Constructor Summary|
|Decrypter() |
| |

|Method Summary |
|protected void|setup() |
| | Initialization method |
|protected void|takeDown() |
| | This method takes down the agent|

|Methods inherited from class jade.core.Agent |
|addBehaviour, afterClone, afterMove, beforeClone, beforeMove, |
|blockingReceive, blockingReceive, blockingReceive, blockingReceive, |
|changeStateTo, clean, doActivate, doClone, doDelete, doMove, doSuspend, |
|doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, |
|getArguments, getContainerController, getContentManager, getCurQueueSize,|
|getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, |
|getProperty, getQueueSize, getState, here, notifyChangeBehaviourState, |
|notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, |
|removeBehaviour, restartLater, restore, restoreBufferedState, run, send, |
|setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, |
|setQueueSize, waitUntilStarted, write |

|Methods inherited from class java.lang.Object |
|clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,|
|wait, wait, wait |

```
|Field Detail|
**** privatekey ****
protected java.security.Key privatekey
================================================================================
**** error ****
protected boolean error
================================================================================
**** mode ****
protected java.lang.String mode
================================================================================
**** secretkey ****
protected javax.crypto.SecretKey secretkey

|Constructor Detail|
**** Decrypter ****
public Decrypter()

|Method Detail|
**** setup ****
protected void setup()
      Initialization method
        Overrides:
            setup in class jade.core.Agent
================================================================================
**** takeDown ****
protected void takeDown()
      This method takes down the agent
        Overrides:
            takeDown in class jade.core.Agent
================================================================================
```

## B.3 Updater.java

```
================================================================================
***** ammaf
Class Updater *****
java.lang.Object
  [extended by ]jade.core.Agent
      [extended by ]ammaf.Updater
  All Implemented Interfaces:
      jade.core.TimerListener, java.io.Serializable, java.lang.Runnable
================================================================================
  public class Updater
  extends jade.core.Agent
This agent generates and updates the hashes file of all the required hosts and
returns to the home host to store them. Note that this agent needs to be
summoned by the Decrypter agent and needs to recieve from it the hashing
```

algotithm, the signer public key and the decrypter secret key encrypted with the home host public key.

See Also:

Serialized Form

---

| Nested Class Summary | | | |
|---|---|---|---|
| (package private) | class | Updater.ChoseNextHostBehaviour | Gets the next host in the list in order to migrate |
| (package private) | class | Updater.FinishHelperBehaviour | Finish the helper lyfecicle. |
| (package private) | class | Updater.FinishUpdaterBehaviour | Sends an encrypted request to the Decrypter agent in order to finish the update |
| (package private) | class | Updater.HackAchieveREResponderBehaviour | A nasty hack to Schedule the FinishHelperBehaviour. |
| (package private) | class | Updater.HashTargets | Hashes the files specifyed in the targets file and generates a hashes file. |
| (package private) | class | Updater.HelperSchedulerBehaviour | Controls and schedules all behaviours depending on the current state |
| (package private) | class | Updater.MigrationBehaviour | Tryes to migrate to the next host |
| (package private) | class | Updater.PackageFilesBehaviour | Packages the newly created hashes file. |
| (package private) | class | Updater.PrepareHelperKeysBehaviour | Prepare the keys needed to encrypt the package and sign it |
| (package private) | class | Updater.PrepareKeysBehaviour | Prepares the cryptographic keys needed to operate on this host |
| (package private) | class | Updater.PrepareReportBehaviour | Tells the Decrypter that the operation has finished when at home and the network hashes update has been finished. |
| (package private) | class | Updater.RecievePackageBehaviour | Recieves the packaged files from the helper |
| (package private) | class | Updater.SchedulerBehaviour | Controls and schedules all behaviours in the main agent depending on the current state |
| (package private) | class | Updater.SendPackageBehaviour | Encrypts signs and sends the package |
| (package private) | class | Updater.SpawnHelperBehaviour | Spawn a helper agent that helps the main agent in the update process |

|Nested classes/interfaces inherited from class jade.core.Agent|
|jade.core.Agent.Interrupted |

|Field Summary |
|protected java.lang.String|homename |
| | |
|protected byte[]|informdecrypterskeyb |
| | |
|protected java.lang.String|nexthostname |
| | |
|java.lang.String|notfoundhash |
| | |
|protected javax.crypto.SecretKey|secretkey |
| | |
|protected java.security.PublicKey|signerpubkey |
| | |
|long|timeout |
| | |

|Fields inherited from class jade.core.Agent |
|AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED,|
|AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN |

|Constructor Summary|
|Updater() |
| |

|Method Summary |
|protected void|afterClone() |
| | Changes the scheduler. |
|protected void|afterMove() |
| | Reregister the required languages and ontologies |
|protected void|beforeClone() |
| | |
|protected void|beforeMove() |
| | |
|protected void|setup() |
| | |
|protected void|takeDown() |
| | Deregisters the agent entry in the df (if the agent|
| |is not a helper). |

|Methods inherited from class jade.core.Agent |
|addBehaviour, blockingReceive, blockingReceive, blockingReceive, |

```
|blockingReceive, changeStateTo, clean, doActivate, doClone, doDelete, doMove,|
|doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, |
|getArguments, getContainerController, getContentManager, getCurQueueSize,    |
|getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject,        |
|getProperty, getQueueSize, getState, here, notifyChangeBehaviourState,       |
|notifyRestarted, postMessage, putBack, putO2AObject, receive, receive,       |
|removeBehaviour, restartLater, restore, restoreBufferedState, run, send,     |
|setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents,        |
|setQueueSize, waitUntilStarted, write                                        |


|Methods inherited from class java.lang.Object                              |
|clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,|
|wait, wait, wait                                                           |


|Field Detail|
**** timeout ****
public long timeout
================================================================================
**** notfoundhash ****
public java.lang.String notfoundhash
================================================================================
**** informdecrypterskeyb ****
protected byte[] informdecrypterskeyb
================================================================================
**** signerpubkey ****
protected java.security.PublicKey signerpubkey
================================================================================
**** secretkey ****
protected transient javax.crypto.SecretKey secretkey
================================================================================
**** nexthostname ****
protected java.lang.String nexthostname
================================================================================
**** homename ****
protected java.lang.String homename

|Constructor Detail|
**** Updater ****
public Updater()

|Method Detail|
**** setup ****
protected void setup()
        Overrides:
            setup in class jade.core.Agent
================================================================================
**** beforeMove ****
protected void beforeMove()
```

```
        Overrides:
            beforeMove in class jade.core.Agent
================================================================================
**** afterMove ****
protected void afterMove()
      Reregister the required languages and ontologies
        Overrides:
            afterMove in class jade.core.Agent
================================================================================
**** beforeClone ****
protected void beforeClone()
        Overrides:
            beforeClone in class jade.core.Agent
================================================================================
**** afterClone ****
protected void afterClone()
      Changes the scheduler. Triggered in the clone when finished cloning
        Overrides:
            afterClone in class jade.core.Agent
================================================================================
**** takeDown ****
protected void takeDown()
      Deregisters the agent entry in the df (if the agent is not a helper).
        Overrides:
            takeDown in class jade.core.Agent
================================================================================
```

## B.4 PackagingTools.java

```
================================================================================
***** ammaf.util
Class PackagingTools *****
java.lang.Object
  [extended by ]ammaf.util.PackagingTools
================================================================================
  public class PackagingTools
  extends java.lang.Object
A simple class that performs the packaging operations part of the AMMAF project
================================================================================

|Constructor Summary                                                          |
|PackagingTools(java.lang.String zippackagefilename, java.lang.String mode) |
|          Constructor method.                                                |


|Method Summary                                                               |
|    void|addToPackage(java.lang.String tobepackagedfilename,                |
|        |java.lang.String packagedfilename)                                  |
```

```
|        |          If this instance is in packaging mode, this method adds a|
|        |new file to the package                                            |
|    void|finishPackaging()                                                  |
|        |          Finishes the packaging process.                          |
|    void|finishUnpacking()                                                  |
|        |          Finishes the unpacking process.                          |
| boolean|hasFile(java.lang.String wantedfile)                               |
|        |          Searches for the desired file in the package.            |
|    void|replaceFiles(java.util.Vector<java.lang.String                     |
|        |[]> desiredreplacements)                                           |
|        |          An operation that copyes the contents of an existing zip |
|        |package to a new one and replaces the desired files.               |
|    void|unpackFileTo(java.lang.String unpackdir,                           |
|        |java.lang.String wantedfile)                                       |
|        |          Searches for the desired file in the package and unpacks |
|        |to the target directory                                            |
|    void|unpackTo(java.lang.String unpackdir)                               |
|        |          Unpacks the contents of the zip package to the given     |
|        |directory                                                          |


|Methods inherited from class java.lang.Object                            |
|clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,|
|wait, wait, wait                                                         |


|Constructor Detail|
**** PackagingTools ****
public PackagingTools(java.lang.String zippackagefilename,
                      java.lang.String mode)
               throws java.lang.Exception
      Constructor method. It starts the packaging.
        Parameters:
            zippackagefilename - The zip file that should be created or
            unpacked
            mode - The execution mode, should be one of the following: "add
            files", "new package", "replace files" or "unpack".
        Throws:
            java.lang.Exception - if there was any error

|Method Detail|
**** replaceFiles ****
public void replaceFiles(java.util.Vector<java.lang.String
[]> desiredreplacements)
                  throws java.lang.Exception
      An operation that copyes the contents of an existing zip package to a new
      one and replaces the desired files. This method should be called only
      once with all the desired replacements and must be called before adding
      new files to the package.
        Parameters:
```

desiredreplacements - the ordered by oldpackagefile name list of string arrays, uses a collator to compare the strings locale wisely. oldpackagefile desiredfile desiredfilepackagename if the parameters desiredfile and desiredfilepackagename are null, the oldpackagedfile is removed from the new package.

Throws:

java.lang.Exception - if ther was any error

---

**** addToPackage ****

```
public void addToPackage(java.lang.String tobepackagedfilename,
                         java.lang.String packagedfilename)
                  throws java.lang.Exception,
                         java.util.zip.ZipException
```

If this instance is in packaging mode, this method adds a new file to the package

Parameters:

tobepackagedfilename - The name of the file that is going to be packaged

packagedfilename - The name that the file will have in the package

Throws:

java.lang.Exception - if there was any error

java.util.zip.ZipException - if the mode is incorrect or the operation has already finished

---

**** finishPackaging ****

```
public void finishPackaging()
                    throws java.lang.Exception
```

Finishes the packaging process. Closes the zipoutputstream and renames the temporaly file to its original name. This method also sets the finished flag to true in order to ban incorrect reutilization of the class.

Throws:

java.lang.Exception - if there was any error

java.util.zip.ZipException - if the mode is incorrect or the operation has already finished

---

**** unpackTo ****

```
public void unpackTo(java.lang.String unpackdir)
              throws java.lang.Exception,
                     java.io.IOException,
                     java.util.zip.ZipException
```

Unpacks the contents of the zip package to the given directory

Parameters:

unpackdir - the target directory where all the files will be unpacked

Throws:

java.lang.Exception - if there was any error

java.io.IOException - if unpackdir does not point to a valid directory

java.util.zip.ZipException - if the mode is incorrect or the

```
            operation has already finished
================================================================================
**** unpackFileTo ****
public void unpackFileTo(java.lang.String unpackdir,
                         java.lang.String wantedfile)
                  throws java.lang.Exception,
                         java.io.FileNotFoundException,
                         java.io.IOException,
                         java.util.zip.ZipException
      Searches for the desired file in the package and unpacks to the target
      directory
        Parameters:
            wantedfile - The file held in the package that must be unpacked
            unpackdir - The target directory where the wanted file must be
            unpacked
        Throws:
            java.lang.Exception - if there was any error
            java.io.FileNotFoundException - if the wanted file was not found
            java.io.IOException - if the unpackdir is not a valid directory
            java.util.zip.ZipException - if the mode is incorrect or the
            operation has already finished
================================================================================
**** hasFile ****
public boolean hasFile(java.lang.String wantedfile)
                throws java.util.zip.ZipException
      Searches for the desired file in the package.
        Parameters:
            wantedfile - The file whose presence in the package is checked
        Returns:
            a bolean whose value is true if the wantedfile has been found,
            false otherwise
        Throws:
            java.util.zip.ZipException - if the mode is incorrect or the
            operation has already finished or if there was a problem reading
            the zipfile
================================================================================
**** finishUnpacking ****
public void finishUnpacking()
                    throws java.lang.Exception
      Finishes the unpacking process. Closes the zipfile and sets the finished
      flag to true in order to ban incorrect reutilization of the class.
        Throws:
            java.lang.Exception - if there was any error
            java.util.zip.ZipException - if the mode is incorrect or the
            operation has already finished
================================================================================
```

## B.5 Hash.java

```
================================================================================
***** ammaf.util
Class Hash *****
java.lang.Object
  [extended by ]ammaf.util.Hash
================================================================================
  public class Hash
  extends java.lang.Object
Digests files using the desired algorithm (default is SHA-256)
================================================================================

|Constructor Summary                                                    |
|Hash()                                                                 |
|          Initializes the Hash digestion, initializing the underlying|
|MessageDigest object                                                   |
|Hash(java.lang.String algorithm)                                       |
|                                                                       |


|Method Summary                                                                 |
| java.lang.String|hashedFilename()                                             |
|                 |          Returns the complete file name of the last file  |
|                 |digested in a Canonical way (i.e standart, with the full   |
|                 |path)                                                      |
| java.lang.String|hashOfFile(java.lang.String filename)                      |
|                 |          Performs the digestion and computes it's hex     |
|                 |string.                                                    |
|              int|length()                                                   |
|                 |          Returns the hash length.                         |
|             void|setAlgorithm(java.lang.String algorithm)                   |
|                 |          Changes the hashing algorithm to the desired one.|


|Methods inherited from class java.lang.Object                          |
|clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,|
|wait, wait, wait                                                       |


|Constructor Detail|
**** Hash ****
public Hash()
     throws java.security.NoSuchAlgorithmException
      Initializes the Hash digestion, initializing the underlying MessageDigest
      object
        Throws:
            java.security.NoSuchAlgorithmException - if the requested algorithm
            doesn't exist
```

```
================================================================================
**** Hash ****
public Hash(java.lang.String algorithm)
     throws java.security.NoSuchAlgorithmException
  Throws:
       java.security.NoSuchAlgorithmException

|Method Detail|
**** hashOfFile ****
public java.lang.String hashOfFile(java.lang.String filename)
                            throws java.io.IOException
      Performs the digestion and computes it's hex string. Note that one Hash
      object can perform several digestions calling this method
        Parameters:
            filename - A string that defines the file that is going to be
            digested
        Returns:
            A string that contains the hash of the filename file in hex format
        Throws:
            java.io.IOException - if there was an error reading the target file
================================================================================
**** hashedFilename ****
public java.lang.String hashedFilename()
                                throws java.io.IOException
      Returns the complete file name of the last file digested in a Canonical
      way (i.e standart, with the full path)
        Returns:
            A string that contains the filename of the last digested file in a
            canonical way
        Throws:
            java.io.IOException - if no file has been digested or there was an
            error getting the path
================================================================================
**** length ****
public int length()
      Returns the hash length.
        Returns:
            An int that means the hash length in bytes
================================================================================
**** setAlgorithm ****
public void setAlgorithm(java.lang.String algorithm)
                  throws java.security.NoSuchAlgorithmException
      Changes the hashing algorithm to the desired one. The desired algorithm
      should exist an should be correct in order to avoid exceptions when
      computing the hash
        Parameters:
            algorithm - A string that defines the file that is going to be
            digested
        Throws:
            java.security.NoSuchAlgorithmException
```

==========

# C Contingut del CD

El CD-ROM adjunt conté tots els fitxers necessaris per tal de poder utilitzar el sistema d'agents ammaf.

En primer lloc, al directori "codi font" s'hi troba el codi font del sistema d'agents en la jerarquia de classes necessària per a que aquest sigui operatiu.

Al directori "documentacio" s'hi troben quatre subdirectoris amb el següent contingut:

- En el director "diagrames" s'hi troben els diagrames en format dia.
- En el directori "javadoc" s'hi troba la documentació javadoc en format html.
- En el directori memòria s'hi troba l'arxiu lyx utilitzat per generar aquesta memòria, i en el subdirectori "figures", els diagrames utilitzats en aquesta memòria.
- En el directori "UML" s'hi troba l'arxiu xmi que conté els diagrames UML emprats en el disseny del sistema i en el subdirectori "TFC_UML" el model UML en format xhtml.

Per acabar, al directori "JAVA" s'hi troba el JDK en la seva versió per a GNU/Linux i els paquets JADE i JADE-S.

# Referències


[1] FAM http://oss.sgi.com/projects/fam.

[2] FingerPrint. Creat per l'empresa 2brightsparks http://www.2brightsparks.com.

[3] JADE http://jade.tilab.com.

[4] JADE-S.

[5] Aglets http://aglets.sourceforge.net/.

[6] FIPA http://www.fipa.org/.

[7] User Mode Linux

[8] VMWARE

[9] ejbca

[10] JSS http://www.mozilla.org/projects/security/pki/jss/

[11] JavaTM Cryptography Extension (JCE) Reference Guide.

[12] JavaTM Authentication and Authorization Service (JAAS) Reference Guide.

[13] Regine Endsuleit & Jackes Calmet. A Security Analisys on JADE(-S) V. 3.2.

[14] Jade Board. Jade Security Guide.

[15] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa & Roland Mungenast. Jade Administrator's Guide.

[16] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco & Giovanni Rimassa. Jade Programmer's Guide.

[17] Jens Krause. Technology Review of Java-Based Mobile Agent Platforms.

[18] Centro de Investigación Científica y de Educación Superior de Ensenada. Consideraciones de seguridad de agentes móviles entre plataformas JADE.

[19] Wayne Jansen & Tom Karygiannis. NIST Special Publication 800-19 - Mobile Agent Security.